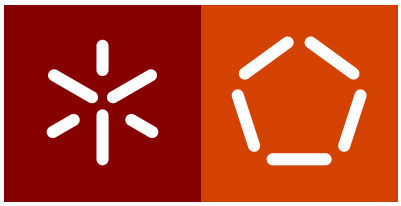
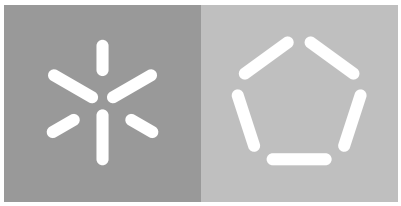


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Henrique Augusto Mateus Gradiz Sobral

Desenvolvimento de uma aplicação web sobre a história dum clube de futebol

Outubro de 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Henrique Augusto Mateus Gradiz Sobral

Desenvolvimento de uma aplicação web sobre a história dum clube de futebol

Dissertação de mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
João Miguel Lobo Fernandes

Outubro de 2017

AGRADECIMENTOS

Gostaria de dirigir os meus sinceros agradecimentos ao professor João Miguel Lobo Fernandes, por ter aceite o meu pedido para ser meu orientador, bem como, pela disponibilidade, apoio, empenho e supervisão que manifestou ao longo de toda a dissertação.

Um muito obrigado a todos os professores que me transmitiram conhecimentos imprescindíveis e se mostraram sempre disponíveis para qualquer questão que pudesse surgir.

Agradeço aos meus colegas de curso, que me apoiaram, e fizeram jus ao companheirismo que existe no meio académico. É verdade que " Da faculdade se levam amigos para a vida!".

À minha esposa por todo o apoio que me deu ao longo desta etapa. Nos momentos em que o mais fácil era desistir, ela ajudou-me sempre a pensar o contrário, tendo sido um apoio fundamental para a conclusão da dissertação.

Aos meus pais e à minha irmã pelas condições e apoio que me ofereceram ao longo do meu percurso académico.

À minha família que foram pilares imprescindíveis durante todo o processo de desenvolvimento desta dissertação. Obrigado por contribuírem tanto para a minha formação como pessoa e como trabalhador.

Agradecer também aos meus amigos e colegas de trabalho que me ajudaram a manter a sanidade durante todo o processo de desenvolvimento desta dissertação.

RESUMO

Qualquer amante do futebol gosta de consultar a história do seu clube, podendo também ver detalhes sobre estatísticas da equipa e de confrontos com outras equipas, sendo que existem poucos clubes que disponham dessa informação.

No que respeita à informação associada a um clube de futebol, muitas vezes a mesma não é fidedigna ou real, existindo muita informação espalhada em diversos sítios, como *websites*, *web services*, jornais, revistas. Não existe uma compilação da informação num único local, dificultando o acesso à comunidade interessada.

O objectivo primordial desta dissertação é a implementação de uma aplicação *web* onde se registará de uma forma centralizada toda a informação referente à história de um clube de futebol, bem como, notícias relacionadas com a equipa, competições, jogos, épocas, jogadores, treinadores, entre outras informações pertinentes no âmbito futebolístico.

Com o intuito de atingir a finalidade supracitada, mostra-se necessária a realização de um estudo sobre alguns projectos já existentes e de carácter semelhante, de modo a conseguir verificar o que é fundamental para que a aplicação se destaque em relação às já existentes no mercado.

Assim, o desenvolvimento deste projecto visa permitir que os adeptos e simpatizantes de futebol consigam consultar determinada informação sobre o clube de uma forma simples e rápida, evitando que o utilizador tenha acesso a informação de veracidade duvidosa.

Para alcançar o produto final que nos propomos desenvolver foi necessário utilizar diversas tecnologias, nomeadamente o *PHP: Hypertext Preprocessor (PHP)* com o auxílio da *framework PhalconPHP*. A escolha destas ferramentas teve por base o desempenho que poderiam oferecer ao produto. Na elaboração da aplicação foi utilizada a última versão do PHP que teve uma melhoria significativa no desempenho.

ABSTRACT

Every football fan appreciates checking his team's history, as well the statistics and previous matches. However, these are only available for a few teams.

Even though part of this intel can be found in several places, like websites, webservices, newspaper, and magazines, often it is not reliable nor real. Also, due to the scattered information, access this data becomes difficult for the interested community.

The primary goal of this thesis is to devise a web application in which all information related to the football club's history, as well as news, competitions, matches, seasons, players, coaches, and additional relevant information to the football community.

To accomplish the abovementioned objective first is essential to go through the state-of-the-art of similar projects. Then, assess the fundamental features to stand out this web application among the existent ones.

Thus, the development of this project aims to deliver fast and simple information to the overall football followers about a specific team, avoiding misguided information.

In order to reach the final product that we propose to develop it was necessary to use several technologies, namely PHP with the help of the PhalconPHP framework. The choice of these tools was based on the performance they could offer the product. In the development of the application was used the latest version of PHP that had a significant improvement in performance.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGENS

ACRÓNIMOS

A

ACL Access Control List.

AJAX Asynchronous JavaScript And XML.

API Application Programming Interface.

C

CLR Commom Language Runtime.

CSRF Cross-Site Request Forgery.

CSS Cascading Style Sheets.

D

DOM Document Object Model.

E

ER Entidade Relacionamento.

F

FCL Framework Class Library.

FTP File Transfer Protocol.

H

HHVM HipHop Virtual Machine.

HMVC Hierarchical Model–View–Controller.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

I

IIS Internet Information Services.

J

JSON JavaScript Object Notation.

M

MVC Model–View–Controller.

O

OOP Object Oriented Programming.

ORM Object Relational Mapping.

OWASP Open Web Application Security Project.

P

PDO PHP Data Object.

PHP PHP: Hypertext Preprocessor.

PSR PHP Standard Recommendation.

R

REST Representational State Transfer.

S

SGBD Sistema de Gestão de Base de Dados.

SMTP Simple Mail Transfer Protocol.

SOAP Simple Object Access Protocol.

SQL Structured Query Language.

SSL Secure Socket Layer.

W

W3C World Wide Web Consortium.

WWW World Wide Web.

X

XML eXtensible Markup Language.

XSS Cross-site Scripting.

1

INTRODUÇÃO

A presente dissertação consiste no desenvolvimento de uma aplicação *web*. O desenvolvimento *web* é uma área em franca expansão, com o surgimento de novas tecnologias, linguagens e *frameworks* a um ritmo elevado. No contexto desta dissertação, justifica-se o desenvolvimento de um estudo sobre as tecnologias utilizadas nesta área, com o intuito de perceber quais poderão ser as tecnologias mais adequadas a usar num dado contexto ou projecto.

O objectivo fulcral na dissertação que nos propomos desenvolver, é a implementação de uma aplicação *web* que permita a um utilizador, entre outras funcionalidades, consultar informação sobre o seu clube, nomeadamente os jogos, as estatísticas dos seus jogadores, o desempenho da sua equipa face aos seus adversários. É ainda importante referir, e com intuito de garantir a segurança e a confiabilidade da informação, que toda a interacção decorrente da aplicação será supervisionada pelo administrador.

1.1 CONTEXTO

Cada vez mais a Internet é um meio de fácil acesso a informação, que de outra forma só podia ser consultada pessoalmente. Torna-se para isso fulcral que, todas as áreas possam disponibilizar a informação que achem adequada e pertinente para os respectivos interessados, necessidade essa que também no ramo do desporto se torna crucial.

O futebol mostra-se um meio de mobilização e aproximação de culturas, sendo por isso importante que a informação futebolística seja de fácil acesso. Sendo o futebol o desporto com mais adeptos em Portugal, estes procuram aceder a toda a informação relevante sobre o mesmo, sendo a procura de informação mais afincada quando existe uma determinada preferência clubística.

Os adeptos querem relembrar-se dos momentos de glória dos seus clubes, existindo por vezes falta de informação sobre determinado evento, como por exemplo qual o onze titular, quem marcou o golo ou quantos adeptos estavam presentes.

As novas tecnologias tem um papel bastante considerável, se não mesmo essencial no dia-a-dia da população. A Internet é um meio de acesso ao mais variado tipo de informação, bastando para isso um simples clique.

Tratando-se de um projecto de desenvolvimento de *software* será necessário definir a abordagem a utilizar, bem como, a arquitectura de *software*. Estes são elementos fulcrais para que um projecto de *software* tenha um desenvolvimento flexível, de modo a existirem diversas melhorias no projecto, assumindo por isso um carácter iterativo. No que respeita à arquitectura de *software* é necessário escolher uma arquitectura que se adeque à implementação de aplicações *web*.

1.2 MOTIVAÇÃO

A escolha deste tema recaiu sobre o especial interesse pelas áreas relacionadas com *web* e com engenharia de *software*. Ao explorar esta área, foi analisada a *framework PhalconPHP*, onde foi possível concluir que aquela permitia aumentar a *performance* das aplicações *web/Websites*, bem como, aos programadores aumentar a sua produção. Outro ponto importante, ponto esse já adquirido durante o primeiro ano de mestrado, relaciona-se com utilização de uma arquitectura de *software*, permitindo que outros programadores consigam facilmente perceber o código implementado e desenvolvido, e facilmente corrigir os erros que possam surgir.

Ao longo da dissertação pretende-se desenvolver uma aplicação *web* que utilize as ferramentas e tecnologias associadas a esta área, com intuito de garantir o seu desempenho e tirar partido de todos os pontos anteriormente referidos.

Outro dos motivos para a escolha do presente tema relaciona-se com o facto de poder aliar duas áreas de interesse, isto é, aliar ao desenvolvimento *web*, o gosto pelo desporto e em concreto pelo futebol. A motivação de trabalhar num projecto que se gosta, torna mais fácil o seu desenvolvimento e consequentemente mais satisfatórios os seus resultados. Fazendo minhas as palavras de Confúcio, " Escolhe um trabalho de que gostes, e não terás que trabalhar nem um dia na tua vida"[1].

1.3 PRINCIPAIS OBJECTIVOS

O objectivo primordial da presente dissertação passa pelo desenvolvimento de uma aplicação *web*, que permita a consulta da história de um clube de futebol, disponibilizando também informações detalhadas sobre os eventos decorridos, mais concretamente os jogos.

De entre os principais objectivos, destacam-se os seguintes objectivos específicos:

- Seguir uma abordagem metódica durante o desenvolvimento prevendo-se adoptar algumas práticas ágeis (e.g., interacção regular com cliente, desenvolvimento iterativo);

- Implementar uma arquitectura de *software* que melhor se adapte às necessidades das aplicações *web*, aprofundando conhecimentos para a sua concepção, bem como, as tecnologias envolventes na sua implementação;
- Realizar um estudo sobre o domínio do tema em que se insere a aplicação *web*, para ser mais fácil satisfazer as necessidades dos utilizadores;
- Implementação da base de dados para suportar toda a informação disponibilizada na aplicação *web*;
- Utilização de uma *framework* para permitir um melhor desempenho da aplicação e estudo de medidas de segurança que concedam uma maior protecção aos seus utilizadores e administradores;
- Realizar cálculos de estatísticas de forma a disponibilizar informação sobre o desempenho do jogador, treinador e plantel;
- Implementar e testar a aplicação *web* ao nível funcional, de *design* e de *performance*.

1.4 Organização da dissertação

A dissertação é constituída por seis capítulos. O primeiro capítulo da dissertação destina-se à contextualização do problema que se pretende tratar nesta dissertação, permitindo ao leitor entender qual vai ser o debate em que assenta toda a dissertação.

No segundo capítulo far-se-à uma descrição de todos os passos, bem como os meios, destinados ao desenvolvimento de uma aplicação *web*, nomeadamente as tecnologias envolvidas e as ferramentas e arquitecturas de *software* que permitem obter um desempenho adequado.

No terceiro capítulo, procurar-se-à relatar todas as técnicas utilizadas para obter os requisitos da aplicação. Neste capítulo serão ainda expostos alguns modelos obtidos, de forma a exemplificar ao leitor como serão desenvolvidos os requisitos.

Em relação ao quarto capítulo serão apresentadas as decisões tomadas para que fossem escolhidas as tecnologias e toda as ferramentas associadas ao desenvolvimento da aplicação. Serão ainda expostos alguns testes feitos à aplicação, devido a ser fulcral testar uma aplicação, de modo a garantir o seu funcionamento.

O quinto capítulo é responsável por demonstrar a aplicação, sendo exploradas algumas funcionalidades, com intuito a dar a conhecer o *software* desenvolvido. Vão ainda ser apresentados os *endpoints* concebidos para a aplicação *web*.

Por fim, no sexto capítulo vão ser apresentadas as conclusões que foram obtidas ao longo da elaboração da dissertação, contudo como é sabido, os produtos de *software* estão em constante mutabilidade e desenvolvimento, por isso serão abordados novas funcionalidades

que poderão vir a ser desenvolvidas futuramente, acautelando assim as possíveis alterações ao nível de novas tecnologias.

2

ESTADO DA ARTE

Este capítulo dedica-se à abordagem necessária para o desenvolvimento de uma aplicação deste âmbito, ou seja, uma aplicação *web*/*Websites*. Neste capítulo constará tudo o que é necessário para a implementação de uma aplicação, desde a definição das funcionalidades à escolha das tecnologias e concepção da aplicação. Nesta fase, procura-se ainda conceder ao leitor da presente dissertação os conhecimentos base sobre esta área, embora de forma um tanto ou quanto superficial. Este capítulo tem ainda o intuito de atingir o rigor de obter a melhor *performance* dos produtos que utilizam as tecnologias associadas ao desenvolvimento *web*.

2.1 PROCESSO DO PRODUTO DE SOFTWARE

O desenvolvimento de um produto de *software*[2] é composto por várias etapas, pois para conceber um determinado *software* é fulcral perceber as ideias/necessidades do cliente, bem como, as escolhas das soluções para desenvolver o que o mesmo pretende.

O primeiro passo é perceber quais as necessidades do cliente e dos utilizadores daquele produto, com o intuito de obter os requisitos da aplicação. Esta fase é fulcral para a concepção de uma solução, visto ser a base da mesma.

Posteriormente é necessário transcrever os requisitos para diagramas, para que depois se faça a implementação do produto.

Por último, a realização de testes é um passo de crucial importância, para validar as funcionalidades dos requisitos, para que posteriormente seja realizado o *deployment* do produto, de forma a permitir o recurso àquela aplicação por parte dos utilizadores.

Nas seguintes secções serão abordados com mais detalhe cada uma das fases do processo de produto de *software*.

2.2 ANÁLISE

A primeira etapa consiste na análise, onde são realizadas várias reuniões com o cliente. No decorrer deste passo é necessário realizar um bom levantamento de requisitos para

que se perceba quais as expectativas do cliente relativamente à aplicação. Numa primeira fase é necessário estudar o domínio em que o produto se vai enquadrar e analisar produtos semelhantes, procurando-se perceber o que introduzir no produto de forma a ser valorizado significativamente em relação aos outros. Posteriormente é necessário ainda identificar as partes interessantes do produto, isto é, qual o público alvo do produto a desenvolver.

Os requisitos dividem-se em dois grupos: os funcionais e não funcionais. Os primeiros são referentes às funcionalidades da aplicação, enquanto que os segundos estão relacionados com a qualidade do *software*, ou seja, usabilidade, segurança, *performance*, entre outros.

Por fim é necessário definir as técnicas de levantamento de requisitos, técnicas essas que serão analisadas nos pontos seguintes.

2.2.1 *Análise de domínio*

A análise de domínio está relacionada com o estudo de soluções já existentes no mercado, bem como, o estudo do domínio em que a aplicação se incide. Esta técnica é muito importante para conseguir identificar os possíveis utilizadores da aplicação e os requisitos do *software*.

O uso desta técnica possibilita, aos analistas, uma recolha mais fácil dos requisitos da aplicação, permitindo ainda detectar as soluções tecnológicas já existentes, de modo a verificar a possibilidade da sua integração no produto em questão.

O estudo sobre outros produtos de *software* permite ainda detectar possíveis falhas, ou seja, ajudando a identificar quais seriam os pontos fortes a implementar no próprio produto, de modo a conseguir destacar-se dos produtos já existente no domínio em estudo.

A figura 1 permite constatar os processos que poderão ser seguidos para conseguir fazer uma eficiente análise de domínio.

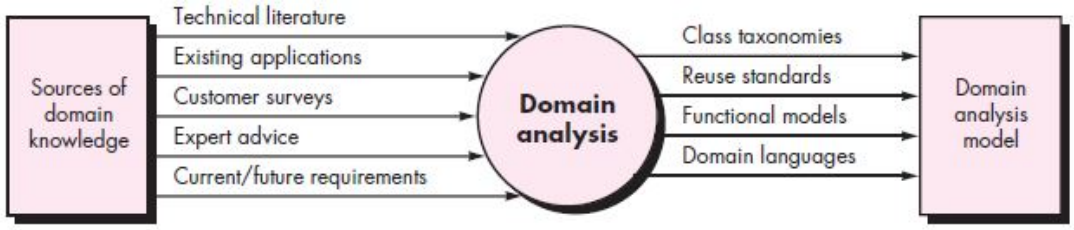


Figura 1.: Análise de domínio - Processo de análise de domínio[3]
(Capítulo 6, p.152)

A (Fig.1) mostra que, na primeira fase, é efectuado o estudo sobre o que existe no domínio em análise, analisando aplicações, livros técnicos ou mesmo reuniões com clientes. Depois de se conhecer o domínio em si, é necessário definir métodos para conceber o modelo de

domínio, que consiste na transcrição do estudo do domínio para um diagrama de modelo de domínio .

2.2.2 *Entrevistas*

A entrevista permite ao analista ter uma aproximação com o cliente identificando as pessoas que deve entrevistar, de modo a obter uma maior qualidade de informação. Esta técnica para ser bem sucedida exige preparação, nomeadamente quando se trata de uma entrevista mais formal, em que, deverá construir-se um guião já com questões elaboradas, de maneira a filtrar-se a informação mais pertinente.

Importa referir que a entrevista pode ser uma conversa mais informal, sendo aconselhada quando o analista conhece bem o domínio em questão. Esta técnica é muito utilizada quando o cliente não tem definido o que pretende, cabendo ao analista prepara a entrevista com algumas questões que ajudem a entender o que o cliente pretende.

Em suma, uma boa entrevista é aquela onde o cliente utiliza a maior parte do tempo. O analista pode interromper o cliente para fazer um resumo daquilo que ele disse, para garantir que a informação obtida era aquela que o cliente queria passar. Caso contrário, poderemos estar perante uma situação de redundância ou de dúvida.

2.2.3 *Personas*

As *Personas* são uma técnica responsável por criar um perfil fictício de uma pessoa, definindo o que ela pretende para o sistema em estudo. A elaboração de *personas* tem como objectivo primordial criar perfis de utilizadores, conseguindo assim associar o tipo de funcionalidades a implementar na aplicação, para que seja possível dar resposta aos mesmos.

As *personas* devem seguir algumas regras, ou seja, deverá ser criado um perfil com dados pessoais, descrever o tipo de pessoa que é, para que seja possível verificar em que tipo de utilizador se pode enquadrar. A informação mais relevante desta técnica são os pedidos que as *personas* enumeram, permitindo ao analista apresentar ao cliente a importância de determinadas funcionalidades que a aplicação deve conter para certo tipo de utilizadores.

2.2.4 *Introspeção*

Esta técnica é utilizada quando o analista tem um conhecimento significativo do domínio em que a aplicação se enquadra. O analista encarna o papel do cliente, definindo ele próprio os requisitos da aplicação, bem como, as estratégias e possíveis falhas que acontecem nas aplicações do domínio em questão.

Contudo é preciso referir que esta técnica deve ser utilizada na primeira fase da análise dos requisitos da aplicação, isto é, no fim do uso desta técnica deverão ser apresentados os resultados ao cliente, garantido que são aquelas as funcionalidades pretendidas para a aplicação.

Este passo é dos mais importantes na concepção de um produto de *software*, uma vez que se não for entendida qual a pretensão do cliente, poderá tal falha ser prejudicial numa fase mais avançada do processo do produto de *software*, defraudando assim as expectativas iniciais daquele.

Concluído o processo de levantamento de requisitos, deverão os mesmo ser escritos, sendo necessário a utilização de frases simples, de modo a evitar ambiguidades e garantir que o requisito é claro e único.

Para a escrita de requisitos é aconselhada a utilização do cartão de *volere*, possibilitando um maior rigor na escrita do requisito e uma maior compreensão do que o requisito representa e deverá realizar (conforme ilustração (Fig.2) que representa o *template* do cartão de *volere*).

Requisito:	**1**	**Tipo de Requisito:**	**2**	**Evento/Caso de uso:**	**3**
Descrição:	**4**				
Fundamentação:	**5**				
Origem/Autor:	**6**				
Critério de Ajuste:	**7**				
Satisfação do Utilizador:	**8**		**Insatisfação do utilizador:**	**9**	
Prioridade:	**10**		**Conflitos:**	**11**	
Materiais de Apoio:	**12**				
Histórico:	**13**				

Figura 2.: *Template* cartão de *volere*

A seguinte lista descreve o que representa cada número constante da figura 2 anterior e a qual passamos explicar:

1. Representa o número do requisito e que por norma é utilizado numa ordem sequencial;
2. Identifica o tipo de requisito, ou seja, se é funcional ou não funcional. Sendo que se pode utilizar uma escala para definir o tipo de requisito (é mais utilizado para os requisitos não funcionais);
3. Referenciar o nome dos casos de uso que estão associados ao requisito, tratando-se de um elemento importante para a concepção dos diagramas de caso de uso;
4. Este campo está associado à escrita do requisito. O que deve ser uma frase simples, de forma a evitar a ambiguidade. É desaconselhado o uso de pronomes possessivos na terceira pessoa (seu, sua, seus, suas) e conjunções (e, ou), sendo que neste caso, perante a existência de uma conjunção, é aconselhável dividir em dois requisitos[2];
5. Descreve o fundamento para a existência do requisito;
6. Identifica o criador do requisito;
7. Descreve um possível cenário que valide o requisito;
8. Representa o nível de satisfação por parte do cliente com a implementação do requisito;
9. Mostra o nível de insatisfação por parte do cliente caso o requisito não seja implementado;
10. Identifica a prioridade do requisito, existindo por vezes uma negociação de requisitos, dependendo do tamanho do projecto ou quando o prazo de entrega é curto;
11. Mostra o conflito com outros requisitos, ou seja, identifica os requisitos que necessitam que o requisito em si esteja implementado;
12. Deste número consta a fonte do levantamento do requisito, ou devendo-se referenciar a origem do requisito, como por exemplo uma entrevista, ou um inquérito;
13. Este campo representa todo o histórico do requisito, desde a criação do requisito até a última alteração (normalmente os requisitos sofrem alterações).

2.3 CONCEPÇÃO

No que respeita à segunda fase do processo do produto do *software*, temos presente a concepção, que é nada mais do que a modelação dos requisitos em diagramas. Esta fase é a passagem de testemunho entre os analistas e os programadores, pois no término desta fase os programadores vão começar a implementar o produto através dos diagramas por aqueles concebidos.

Os diagramas vão ser fundamentais para os programadores entenderem o fluxo a implementar para o cumprimento da especificação do requisito. Esta abordagem vai permitir que os programadores consigam implementar o produto com mais brevidade e melhor qualidade. Esta fase também é importante, para a definição das tecnologias e arquitectura de *software* a utilizar, de modo a enquadrar-se com as necessidades e especificações que o cliente pretende, estando aqui patente a importância dos requisitos não funcionais para estas escolhas.

Para exemplificar os diagramas, e com o intuito de dar uma percepção mais simples, foram elaborados alguns diagramas sobre um cenário superficial, relacionado com clientes e as suas respectivas agências bancárias.

2.3.1 *Modelo de domínio*

O modelo de domínio permite descrever de uma forma mais sintética o domínio que foi analisado, ou seja, descreve os processos, regras ou lógicas que se devem seguir para implementar a aplicação. Este diagrama ajudará na concepção da aplicação em si, pois é onde se começa a definir a arquitectura da aplicação.

A concepção do modelo de domínio vai ser fundamental para o desenvolvimento do diagrama de classes e do modelo de entidade-relacionamento.

A figura 3 mostra o exemplo elencado no supracitado ponto 2.3.1 .

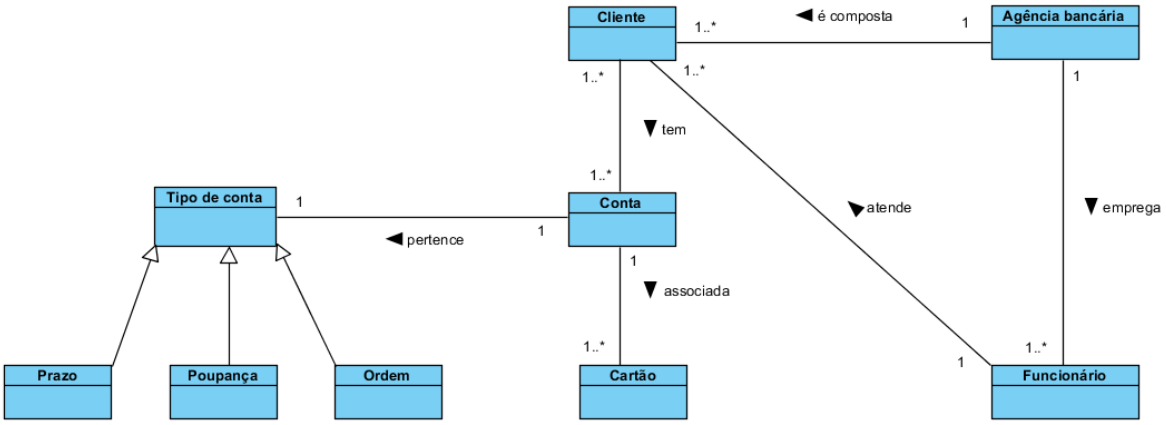


Figura 3.: Modelo de domínio de clientes de agências bancárias

Neste modelo de domínio, é possível identificar diversos aspectos:

- O cliente pode ter várias contas, sendo que a cada uma será associada um tipo de conta, com um ou vários cartões associados;
- A agência bancária é composta por vários clientes e têm vários funcionários dessa mesma agência. Cada funcionário atende diversos clientes;
- Uma conta pode ser constituída por um ou mais clientes.

Embora estejamos perante um exemplo simples e com poucas regras associadas e pouco detalhadas, é possível comprovar a importância de um modelo de domínio.

Assim é possível concluir que para que o diagrama seja mais completo é necessário realizar uma extensa análise, de modo a entender todas as regras e lógica de negócio associados ao domínio em questão.

2.3.2 *Diagrama de classes*

O diagrama de classes é fundamental para qualquer aplicação que utilize o paradigma de OOP, pois é aqui que se define a estrutura de cada classe. Neste diagrama (Fig.4) também se definem os atributos, métodos, relações e o tipo de classes a ser utilizados.

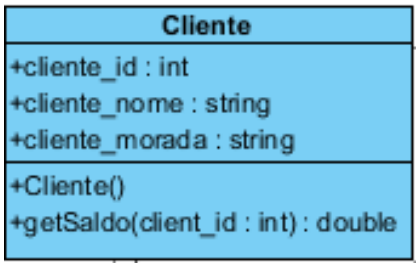


Figura 4.: Estrutura de uma classe

Na figura 4 é possível verificar a estrutura de uma classe. A primeira área define o nome da classe, a segunda área define os atributos da classe e por fim, no que respeita a terceira área, estão definidos os métodos da classe. É necessário referir que tanto nos atributos como nos métodos é possível definir a visibilidade, bem como, o tipo de dados que são retornados, sendo que a visibilidade pode assumir os seguintes valores:

- **Public +** : Qualquer classe pode aceder aos métodos/atributos;
- **Private -** : Só a própria classe é que tem permissão para aceder aos métodos/atributos;
- **Protected #** : Permite o acesso aos métodos/atributos da própria classe, bem como, das classes filhas (Generalização/Herança);
- **Package** ~: Só podem ser acedidos por classes do mesmo *package*;

Ao diagrama de classes estão subjacentes algumas particularidades, nomeadamente a multiplicidade, isto é, o que estabelece a relação entre as classes, de forma a perceber por exemplo quantos objectos a classe A pode ter da classe B ou vice-versa, tais como as seguintes:

- **1 - 1**: Define que cada objecto da primeira classe só pode estar ligado a um único objecto da segunda classe;
- **1 - 1..*** : Define que cada objecto da primeira classe terá pelo menos um ou muitos objectos da segunda classe;
- **1 - 0..*** :Esta multiplicidade é muito semelhante a anterior, no entanto a diferença é que a primeira classe pode não estar ligada a nenhum objecto da segunda classe;
- **1..* - 1..*** :Define que um ou mais objectos de cada classe pode estar relacionado com um ou mais objectos da outra classe;
- **Herança**: A herança é utilizada quando se pretende que uma classe utilize métodos da classe pai, mas que implemente os seus próprios métodos, de modo a diferenciar-se das outras classes filhas caso existam. A herança é representada por um triângulo na classe pai.

Na figura 5 é possível visualizar o diagrama de classes referente ao caso em estudo.

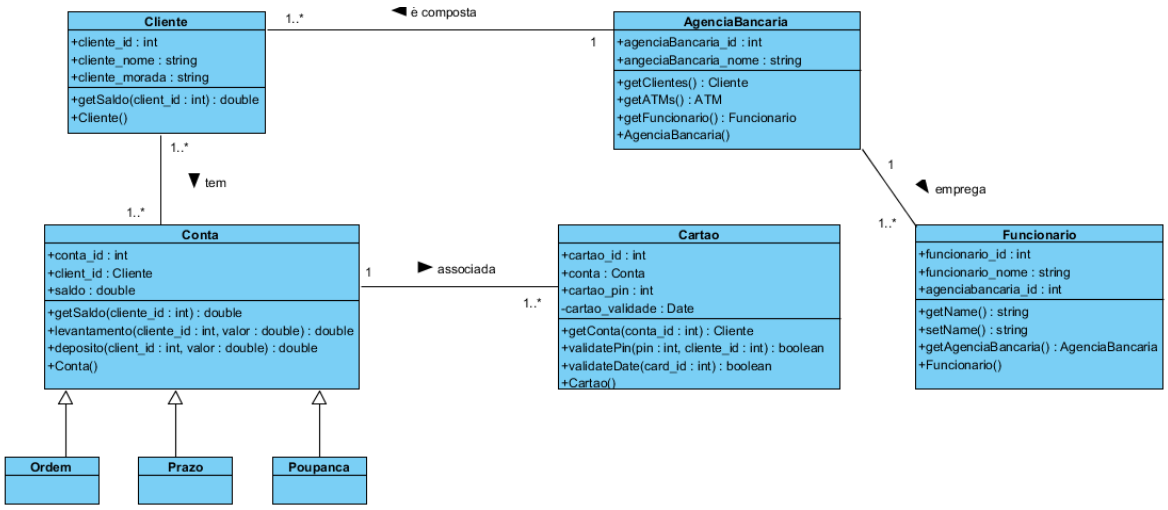


Figura 5.: Diagrama de classes

O diagrama acima representado (Fig.5), permite ao programador estruturar as classes, já com os dados e métodos associados a cada uma delas. Na maioria das associações de classes foram utilizados algumas multiplicidades, nomeadamente a herança. Um exemplo disso é a classe Conta que tem várias classes filhas, que herdam todos as propriedades da classe pai, mas os métodos em si têm outra lógica.

2.3.3 *Modelo ER*

O modelo ER descreve a estrutura da informação, ou seja, a forma como a base de dados tem de ser organizada e inter-relacionada. No que respeita a este modelo, uma entidade representa uma tabela que terá associados campos e o seu tipo de dados. Como estamos perante uma base de dados relacional, é necessário utilizar chaves primárias (*Primary Key*) e estrangeiras (*Foreign Key*), de modo a estabelecer uma relação entre entidades. A chave primária é associada a um ou mais atributos de uma tabela, sendo que o valor da mesma não pode ser utilizada noutro registo da mesma tabela. No que concerne a chave estrangeira, é um atributo de outra tabela que se refere a uma chave primária, com o intuito de estabelecer a relação entre tabelas.

A relação das tabelas também segue umas condições, tal como o diagrama de classes, seguidamente são enumeradas:

- **1 - 1**: Uma relação de 1-1, representa que um registo da Tabela A está relacionado apenas com um único registo da tabela B;

- **1 - N**: "Uma relação 1-N ocorrem quando um registo de uma tabela A está vinculado com muitos registos da Tabela B"[4](Tradução nossa). Com base nas palavras do autor *Robin Nixon* podemos afirmar que no modelo seguinte (Fig.6) é possível verificar que um determinado cliente pode ter um ou mais registo na tabela conta.

- **N - M**: O relacionamento N-M possibilita que muitos registos da Tabela A estejam relacionados com muitos registos da Tabela B, para esta relação terá que ser criada uma tabela intermédia, que irá ser constituída por uma chave composta (são as chaves primárias de cada uma das tabela originais).

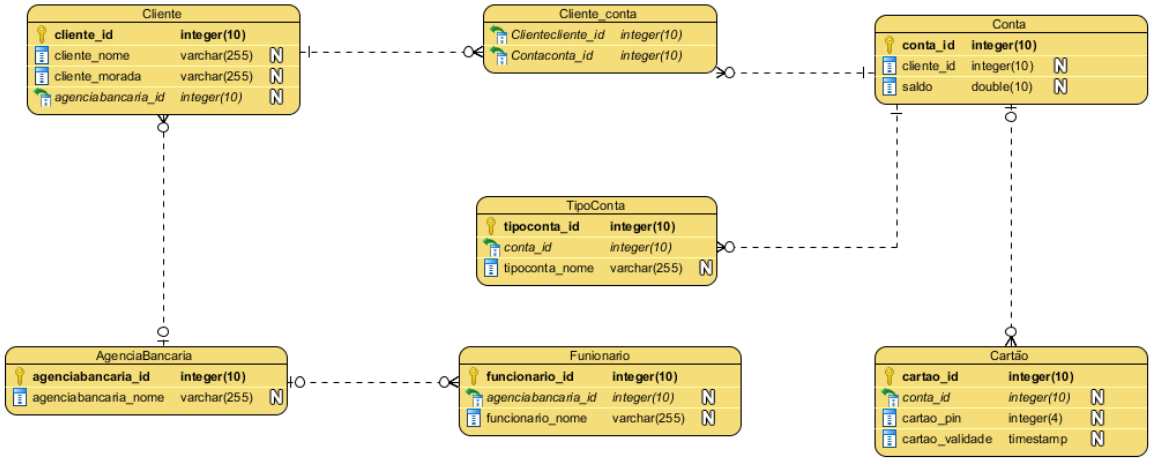


Figura 6.: Modelo de ER

Na figura 6 é possível verificar o modelo ER, modelo esse em que foi utilizada a relação 1 - N e N - M. Através da análise do modelo acima (Fig.6) elencado é possível verificar que na relação entre as entidades cliente e conta (relação N - M) foi criada uma tabela intermédia. Essa tabela será responsável por guardar todas as contas associadas a um determinado cliente e vice-versa.

2.3.4 *Casos de uso*

O diagrama de casos de uso é um diagrama fulcral, pois descreve quem são os actores da aplicação, bem como, as funcionalidades que a aplicação irá disponibilizar aos seus utilizadores. Podemos considerar que um caso de uso é um cenário, que associa actores a esse mesmo cenário, e que por sua vez demonstra o que determinada funcionalidade precisa para ser realizada .

No que respeita às associações dos casos de uso, temos dois tipos *include* e *extend*. *Include* define que determinado caso de uso precisa de outro caso de uso para realizar determinada

operação. Ao *extend* compete adicionar um determinado comportamento a um caso de uso base.

A figura 7 representa o diagrama de caso de uso, onde é possível verificar alguns eventos associados ao cliente.

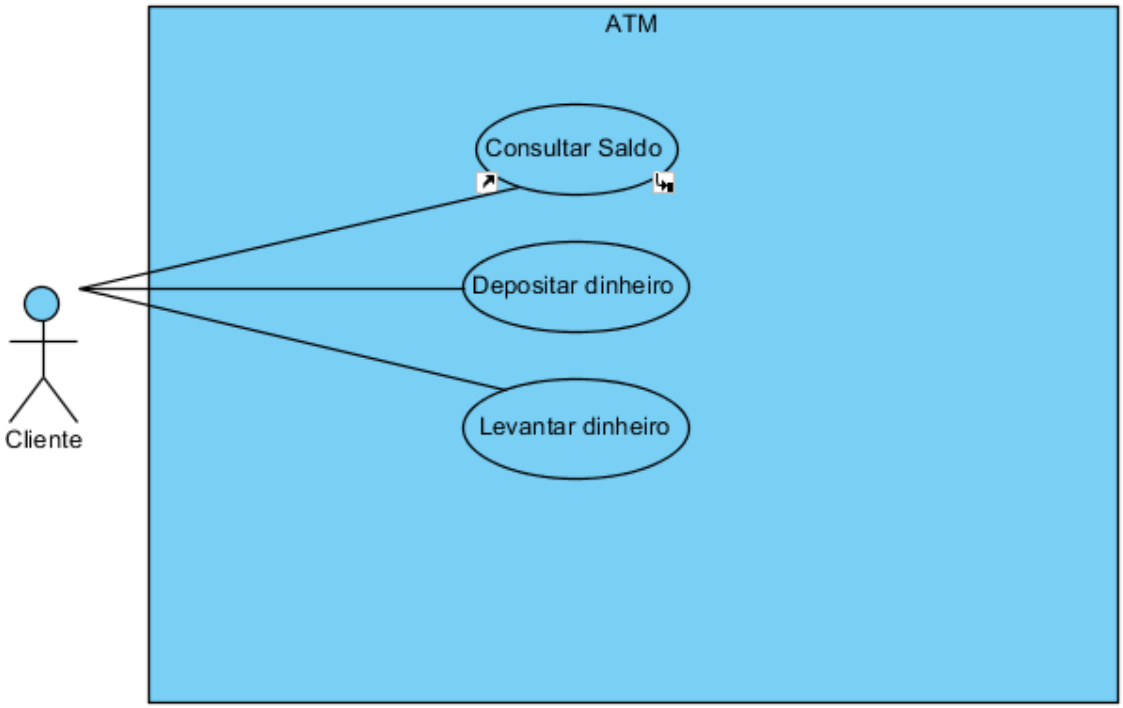


Figura 7.: Diagrama de caso de uso do ATM

2.3.5 *Diagrama de actividades*

O diagrama de actividades estabelece o fluxo que deve ser seguido para determinado caso de uso, definindo todos os passos cuja a obrigatoriedade deve ser verificada para que determinada operação seja concluída.

Este diagrama permite ainda prever e definir todos os cenários, em caso de erro ou insucesso, conferindo ao programador a possibilidade de implementar todos os cenários possíveis de sucesso ou insucesso.

Na figura 8 podemos verificar o fluxo necessário para que um cliente consiga levantar dinheiro. A bola preta significa o começo da operação, os balões azuis são considerados um passo no fluxo. Quanto ao losango define uma decisão, nomeadamente um ciclo condicional (*if*), que consoante o resultado, vai alterar o processo. Trata-se de um ciclo importante para garantir os casos em que o processo não corre como esperado.

No diagrama representado (Fig.8) é garantido o fluxo necessário para a operação ser realizada com sucesso.

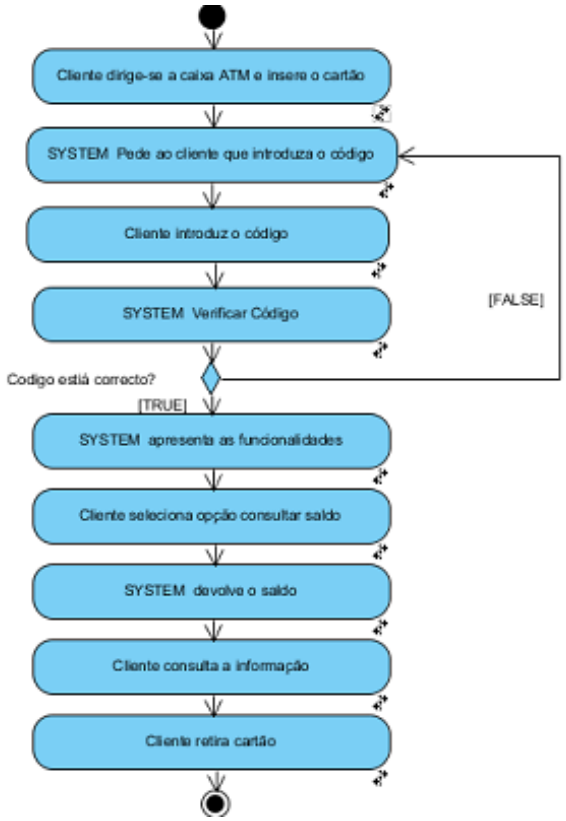


Figura 8.: Diagrama de actividades - (Consultar saldo)

2.3.6 *Diagrama de sequência*

Em relação ao diagrama de sequência abaixo representado (Fig.9) importa referir, que não é nada mais do que a interacção entre os objectos, ou actores com o sistema, identificando-se os métodos que estão associados ao caso de uso em questão.

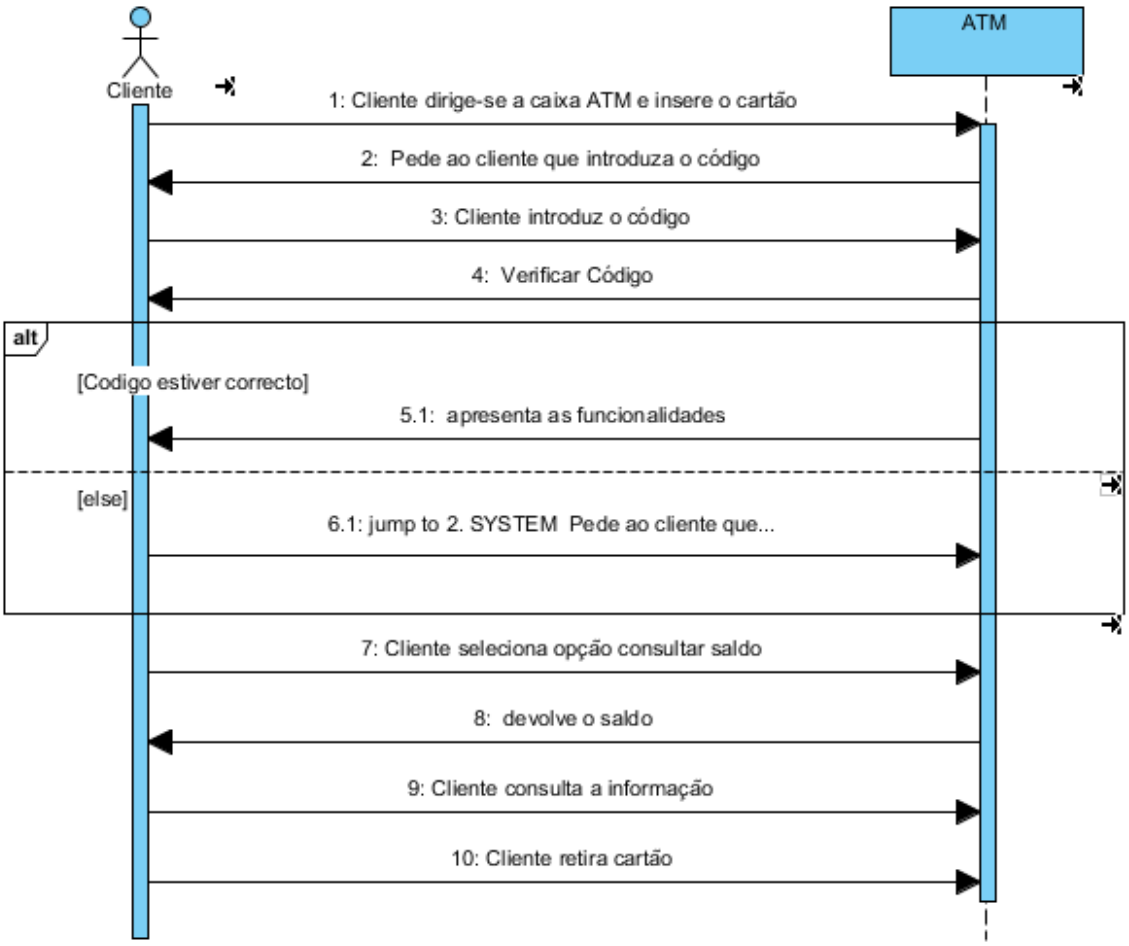


Figura 9.: Diagrama de sequência - (Consultar saldo)

Estamos perante um diagrama de tecnicidade acrescida, visto ser uma comunicação entre os objectos e o sistema, possibilitando o uso dos métodos associados a cada objecto. Este diagrama complementa o diagrama de actividades já anteriormente mencionado (Fig.8) e permite ao programador ter uma visão mais abrangente sobre o que tem que desenvolver.

2.3.7 *Arquitectura de software*

A arquitectura de *software* consiste na delimitação dos elementos integrantes do *software*, as suas características e a sua relação com outros *softwares*.

A arquitectura de *software*, pode ser equiparado ao projecto de uma casa mas a nível de *software*, pois demonstra a estrutura do programa e dos seus componentes para que os programadores consigam entender como conceber o seu código, de modo a seguir as regras da arquitectura e consequentemente tirar partido dessa arquitectura.

A definição de uma arquitectura de *software* é feita através dos atributos de qualidade associados ao mesmo, ou seja, é definida pelos requisitos não funcionais. A implementação de arquitecturas de *software* proporciona grandes vantagens tais como :

- Possibilita a divisão do produto em módulos, permitindo reutilizar código já implementado, o que reduz os erros no *software*, bem como, o tempo da concepção do produto;
- Estabelece uma estrutura para o *software*, ajudando os programadores a compreender de uma forma mais simples a aplicação;
- Permite identificar os componentes importantes para que a aplicação funcione, ajudando ainda a arranjar formas redundantes para possíveis falhas.

2.3.7.1 *Cliente e Servidor*

Esta arquitectura de *software* é utilizada quando um cliente pretende aceder a determinado recurso que se encontra no servidor. No caso de um *Website*, o servidor irá ter a aplicação alojada, com todos os recursos necessários para o seu funcionamento. No que respeita ao cliente terá que aceder a um *browser*, de modo a conseguir interagir com a aplicação, existindo uma comunicação entre o *browser* do cliente e a aplicação.

Sempre que o cliente fizer um determinado *request* à aplicação, estará a solicitar recursos ao servidor, ou seja, cabendo àquele responder à solicitação do cliente. Esta arquitectura tem diversas vantagens como a segurança e *performance* da aplicação uma vez que é possível dividir as complexidades por diversos servidores, de modo a que a mesma seja garantida, e permite evitar uma possível sobrecarga no servidor.

2.4 IMPLEMENTAÇÃO

No terceiro passo (a implementação), a equipa de desenvolvimento constrói o sistema de *software*. Seguidamente vão ser explorados temas que estão relacionados com o desenvolvimento de um produto.

2.4.1 *Metodologias de desenvolvimento*

As metodologias de desenvolvimento são abordagens que permitem a uma equipa de desenvolvimento seguir determinados passos para chegar ao objectivo final. O principal objectivo das metodologias é desenvolver *software* com qualidade, de uma forma mais organizada e rápida permitindo diminuir o número de erros existentes no produto. Este processo tira ainda partido de uma maior comunicação entre os envolventes no projecto, fomentando a entreajuda da equipa e reflectindo-se numa maior produtividade.

Existem dois tipos de metodologias de desenvolvimento as ágeis e as tradicionais. No que concerne às metodologias tradicionais, devem ser utilizadas quando já existe um vasto conhecimento sobre os requisitos a implementar em determinada aplicação. As metodologias ágeis têm como foco principal corrigir problemas que eram originados antes do seu aparecimento, ou seja, a falta de adaptabilidade ao aparecimento de novos requisitos.

No que respeita à escolha das metodologias de desenvolvimento, a sua escolha deve ser realizada de acordo com o projecto e com os recursos disponíveis para determinado projecto. O número de recursos, o orçamento para o desenvolvimento do produto ou o prazo para a entrega do produto são alguns dos factores a ter em conta na decisão.

Seguidamente será explicada ao pormenor uma metodologia de cada tipo, das aplicadas na concepção do *software*.

2.4.1.1 *Cascata*

A cascata é um modelo de processo de *software* muito utilizado, nem sempre sendo perceptível o uso desta metodologia. Esta metodologia está divida nas seguintes fases:

- **Análise**: Realização do levantamento de requisitos;
- **Concepção**: Concepção de diagramas que ilustram os requisitos da aplicação, bem como, os passos a seguir para implementar as funcionalidades;
- **Implementação**: Transformar os diagramas em código, de modo a implementar as funcionalidades no produto;
- **Testes**: Proceder a testes sobre as funcionalidades implementadas.

Esta metodologia tem alguns aspectos positivos, como por exemplo o desenvolvimento de modo faseado, apenas se avançando para a fase seguinte após o término da anterior. Contudo esta metodologia existe mudanças e testes exaustivos que são realizados no final da implementação.

Em suma, esta abordagem deve ser utilizada quando se dispõe de poucos recursos para o desenvolvimento do produto, devendo contudo salientar-se a baixa complexidade como um ponto positivo, uma vez que estas fases estão naturalmente presentes no desenvolvimento de um produto de *software*.

2.4.1.2 *SCRUM*

Esta metodologia é utilizada para dividir os recursos em várias equipas, de modo a obter melhores resultados, sendo que as tarefas para cada equipa/elemento vão ser associadas a um *sprint*. Um *sprint* define o que é preciso conceber e qual o prazo para o fazer.

Um dos ponto chave para utilização do *SCRUM* são as reuniões realizadas frequentemente tanto com o cliente como com os elementos das equipas.

As reuniões são agendadas conforme o gestor do projecto decidir, e é feito um ponto da situação das tarefas, com o intuito de analisar o estado final do *sprint*.

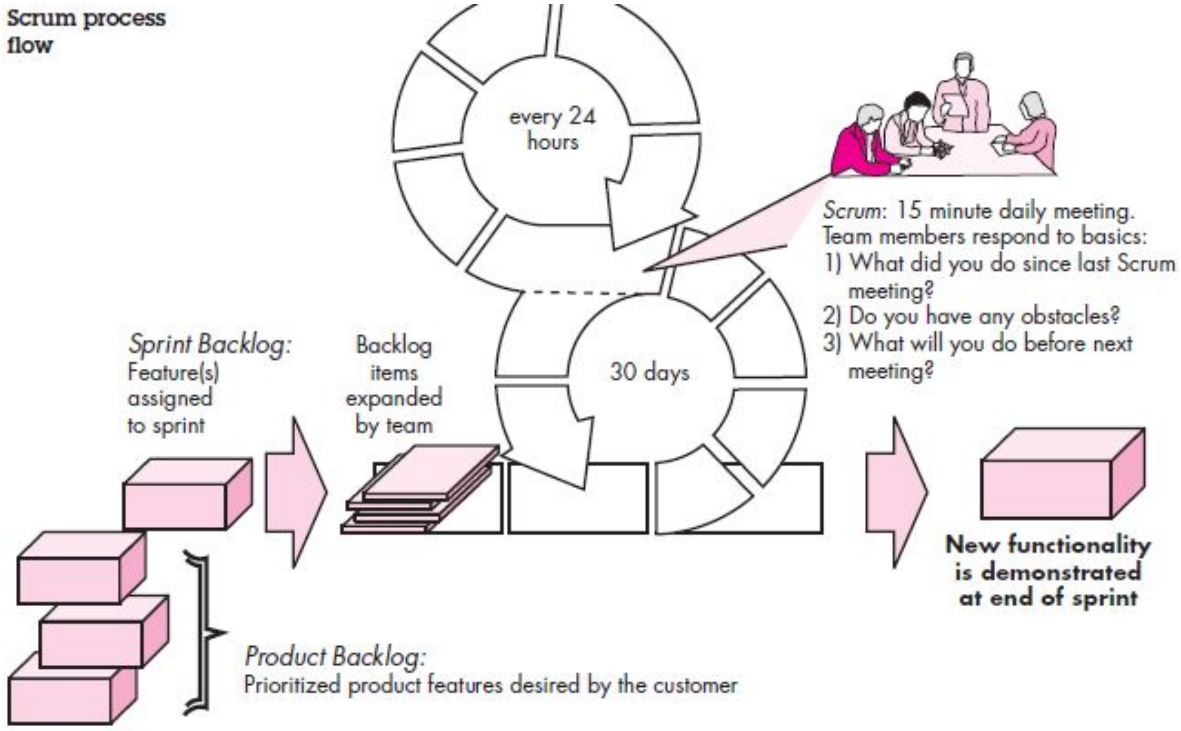


Figura 10.: SCRUM - Fluxo de processo
[3](Capítulo 3 - p.83)

Como é possível analisar na figura 10 existem reuniões todos os dias onde é realizado o ponto de situação das tarefas. O *sprint backlog* refere-se as tarefas que foram incluídas no *sprint* que vai decorrer durante estes 30 dias, não sendo obrigatório *sprint* deste prazo, sendo mais recorrente os *sprints* de 15 dias (2 semanas).

O *SCRUM* é uma metodologia aconselhável quando se dispõe de uma equipa até 8-9 elementos envolvida num projecto, pois facilita a divisão de tarefas, bem como, a interacção entre os elementos da equipa de desenvolvimento. É necessário referir que esta metodologia permite uma maior aproximação ao cliente, e que por vezes é o cliente que ajuda a definir os *sprints*.

No fim do *sprint* é preciso verificar se as funcionalidades foram realizadas com sucesso, ou seja, se cumpre com o que o gestor de projecto ou cliente solicitou (*sprint retrospective* e *review meeting*).

2.4.2 *Plataformas*

As plataformas na área das tecnologias de informação é o local onde a aplicação é executada, de modo a que haja uma operacionalização para os seus utilizadores. Nesta secção 2.4.2 serão abordadas dois tipos de plataformas de aplicações, onde vão ser explorados alguns prós e contras de cada uma.

2.4.2.1 *Plataforma web*

É necessário evidenciar a importância da plataforma *web* para o desenvolvimento *web*, uma vez que esta possibilita aos programadores o desenvolvimento dos seus projectos sem requerer da instalação de *software*. Com o intuito de aumentar a qualidade dos produtos *web* face aos produtos de *software* nativos, houve a necessidade de lançar novas medidas que permitissem tornar a *web* numa plataforma, surgindo a *web* 2.0.

Tendo como base TIM O'REILLY[5] o termo *web*2.0 consiste na mudança para uma Internet como plataforma abrangendo todo os dispositivos conectados. Entre outras, a regra mais importante é desenvolver aplicativos que aproveitem os efeitos de rede para se tornarem optimizados quanto mais são usados pelas pessoas, aproveitando-se a inteligência colectiva.[5]

A *web* 2.0 tornou as soluções *web* mais robustas e mais atractivas para os utilizadores e programadores, pois permitiu que várias linguagens de programação interagissem entre si. Patente a esta evolução está o dinamismo concedido aos *Websites* actuais deixando estes de ser estáticos e concedendo-lhes um *design* desenvolvido.

São características da *web*2.0 as interfaces ricas e fáceis de utilizar, a facilidade de armazenamento de dados e a criação de páginas *online*, permitindo ainda o acesso de vários utilizadores à mesma página e a respectiva edição de informação. A utilização de plataformas *web* permite aos utilizadores o uso das aplicações *web* desde que possuam acesso ao servidor HTTP, possibilitando-lhes assim uma grande portabilidade, contudo tem alguns problemas associados à segurança, que serão abordados posteriormente.

Sendo esta uma área em constante mutação, onde nada é eterno, ouvem-se rumores acerca da *web*3.0.

2.4.2.2 *Plataforma Nativa*

Nas plataformas nativas o *software* opera apenas sobre o nosso computador, tendo o cliente que estar ligado a uma determinada rede local, o que consubstancia numa portabilidade nula, contudo, do ponto de vista da segurança não se encontra tão exposto devido ao facto de estar apenas disponível na rede ou no computador que está instalado.

As aplicações nativas são mais utilizadas quando se trata de contabilidade ou casos de estudo relacionados com a ciência, sendo adequado para aplicações onde se pretende a

máxima segurança e quando o número de utilizadores é muito restrito, por exemplo uma empresa ou um grupo pequeno de utilizadores. São preferenciais quando necessitam de uma grande capacidade de processamento.

2.4.3 *Aplicações web e Website*

Tal como em todos os projectos, há uma essencialidade na definição do seu objectivo, ou seja, no "bem" ou resultado final que se propõe atingir. Assim, e em primeira instância, é necessário definir o que se entende por aplicação *web* e *Website* com o intuito de clarificar a diferença entre esses dois termos.

Uma aplicação *web* pode ser considerada uma aplicação desenvolvida em tecnologia *web* sendo executada do lado do servidor e permitindo que o cliente aceda à mesma através do *browser*. Um *Website* é um aglomerado de ficheiros *HyperText Markup Language (HTML)* que constam no mesmo endereço, sendo que os *Websites* têm uma informação mais estática, ao contrário das aplicações *web*, sendo estas quase na sua totalidade dinâmicas.

Com o padrão de desenho MVC, que será posteriormente abordada na secção 2.4.6.2, conseguiu-se ter uma junção de aplicações *web* e *Website*, ou seja, ter a estrutura de endereço de um *Website* mas ter a informação de forma dinâmica através da base de dados. Com o dinamismo concedido às aplicações *web/Website* estas tornaram-se mais atractivas, em relação às já existentes.

2.4.4 *Desenvolvimento web*

O desenvolvimento *web* incide na expansão de soluções sejam elas *Websites* ou aplicações *web*, consubstanciando-se na divisão de dois módulos: *backend* e *frontend*. O *backend* é referente ao administrador, sendo possível editar conteúdo, entre outras opções, e o *frontend* destina-se aos utilizadores, permitindo o seu acesso para visualizar e interagir com o *Website*. No que respeita ao desenvolvimento *web* é necessário a utilização de diversas ferramentas e tecnologias. Para ser elaborada a estrutura do *Website* deverá ser utilizado o HTML e para o *design* tem que ser utilizado o *Cascading Style Sheets (CSS)*. Pode ainda ter a contribuição das tecnologias *JavaScript* e *jQuery* que possibilitam o manuseamento dos elementos da estrutura da página do *Website*.

Por fim, temos a linguagem PHP, *ASP.NET* entre outras que correm no servidor, e que funcionam de intermediário entre o cliente e servidor. Assim, permite-se a realização de transações ao servidor da base de dados, de forma a recolher informações para apresentar no lado do cliente, no caso da informação ser solicitada pelo mesmo. Destaque ainda para a tecnologia *Asynchronous JavaScript And XML (AJAX)* que permite solicitar informação ao servidor, tendo como vantagem a não realização de um *load* da página total, o que permite

um aumento do desempenho do *Website*, e possibilita uma interacção mais agradável ao utilizador. Sendo ainda necessário salientar a utilização de *web services* que permitem a integração de aplicações móveis. Os *web services* são fundamentais para comunicar com outras aplicações, definindo *endpoints* para que outras aplicações consigam comunicar.

2.4.5 *Servidor HTTP*

O servidor HTTP é responsável por correr as aplicações que estão alojadas no mesmo, ou seja, é a peça chave para o funcionamento de qualquer solução *web*, cabendo-lhe dar a resposta aos pedidos realizados pelo cliente.

O funcionamento do servidor é resumido a um pedido de um cliente, que transmite ao servidor o *url* a que pretende aceder, sendo o servidor responsável por saber qual é a aplicação respectiva e permite aceder a todos os recursos necessários para responder ao cliente.

No que respeita a aplicações que utilizem base de dados, o servidor deve fazer uma comunicação com o servidor da base de dados, de modo a conseguir fazer os pedidos que a aplicação necessita para completar a resposta a fornecer ao servidor e para posteriormente ser enviada ao utilizador. Em termos de segurança os servidor HTTP também são responsáveis por utilizar o SSL que permite a codificação dos conteúdos que são por exemplo introduzidos pelos utilizadores, evitando que sejam captados.

Existem dois tipos de servidores:

- **Dinâmicos**: São os servidores que utilizam uma base de dados, ou seja, antes de enviar os ficheiros necessários para responder ao cliente, têm que consultar a base de dados e actualizar os ficheiros com os valores devolvidos pela base de dados;
- **Estáticos**: São os servidores que apenas enviam os ficheiros associados ao pedido sem ter que os alterar;

Estes servidores ainda oferecem várias funcionalidades, de modo a aumentar o desempenho das aplicações neles alojados, nomeadamente o uso de mecanismos de *cache* e balanceamento de carga (quando sobrecarregado envia os pedidos para outros servidores). O servidor HTTP tem a capacidade de executar *scripts* em diversas linguagens, tais como PHP, *ASP.NET* entre outras. Os servidores HTTP mais utilizados para aplicações e soluções *web*[6] são o *Apache*[1], *Nginx*[2] e o *Internet Information Services (IIS)*[3] da *microsoft*.

1 Página web: https://www.apache.org/
2 Página web: https://www.nginx.com/
3 Página web: https://www.iis.net/

2.4.6 *Design patterns*

Os *design patterns* ou padrões de desenho em português, são cruciais para definir a estrutura do código implementado, ou seja, pode ser considerado como uma norma a seguir na codificação do produto. A escolha de *design patterns* é importante pois permite aos programadores criar um código organizado e menos complexo, facilitando a percepção do mesmo por parte de novos elementos. Importa definir que o uso dos mesmos permite aumentar a modularidade da aplicação, bem como, o uso de soluções que aumentam a *performance* da aplicação.

Seguidamente serão exploradas alguns *design patterns*, de modo a analisar a vantagens do uso dos mesmos.

2.4.6.1 *Big ball of mud*

Este padrão é muito utilizado em *softwares* com baixa complexidade, sendo necessário salientar que esta técnica deve ser sempre evitada.

Estamos perante o uso da mesma quando nos deparamos por exemplo com um *software*, onde o seu código fonte está todo num único ficheiro. O produto até pode cumprir o que o cliente pediu, só que imagine-se que existe um problema naquele código, a identificação e correcção do erro será um processo moroso e trabalhoso. Outro ponto essencial para nunca se utilizar este padrão é que a sua modularidade irá ser nula, pois o código está todo no mesmo ficheiro.

Em suma, este padrão é de evitar, uma vez que a maior parte dos produtos de *software* estão em constante mutabilidade e este padrão não acautela essa evolução.

2.4.6.2 *MVC*

O *design pattern* MVC [7] permite dividir a complexidade do sistema por vários sistemas/camadas. É muito utilizado quando se está perante um problema com alguma complexidade e garante concorrência uma vez que as camadas vão ser independentes umas das outras e com isso é possível estender a aplicação com novas funcionalidades. Este padrão tira partido da facilidade de acrescentar novas funcionalidades, e claro, na reutilização de código facilitando assim a concepção e a correcção de erros.

O MVC está dividido em três camadas são elas as seguintes :

- ***Model*** : O modelo é responsável pelo armazenamento dos dados, ou seja, está relacionado com a base de dados. Esta camada é fundamental para solicitar os pedidos das acções desencadeadas pelo utilizador.

- ***View***: Esta camada é responsável por apresentar os dados enviados pelo *controller* para que os utilizadores tenham a possibilidade de os visualizar.

- ***Controller***: Permite executar todas as acções que são solicitadas pelos utilizadores através das *views*. Consoante a acção solicitada pelo utilizador, o *controller* vai responder a esse pedido, acedendo ao modelo para consultar toda a informação que necessita, com a finalidade de enviar a resposta ao pedido do utilizador para a *view*.

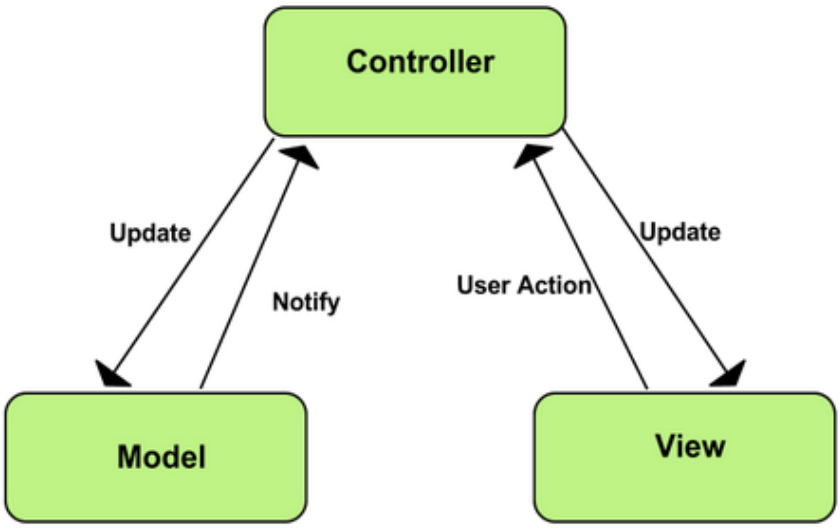


Figura 11.: Arquitectura de *software* MVC estrutura [8]

2.4.6.3 Hierarchical Model–View–Controller (HMVC)

O HMVC é uma aplicação do MVC mas em forma hierárquica, ou seja, pode ser utilizado em aplicações que utilizem *backend* e *frontend*. Essa utilização permite que se tenha os dois módulos da aplicação, utilizando o mesmo padrão de desenho, mas ainda assim serem independentes um do outro.

Em termos de manutenção do produto de *software* torna-se mais fácil pois reduz a complexidade, por estarem separados.

É ainda necessário referir que esta abordagem permite aumentar a extensibilidade de qualquer *software*, bem como, o aumento da organização do produto, permitindo por exemplo agrupar certas funcionalidades relacionadas, de modo a criar um módulo da aplicação.

2.4.7 *Tecnologias*

As tecnologias a nível de aplicações *web*, dividem-se em três tipos:

- ***Client side*** : *Client side* são todas as linguagens que operam no lado do cliente, ou seja, tudo o que é corrido ou manuseado pelo cliente;

- ***Server side*** : No que respeita ao *server side* temos presente as tecnologias que correm do lado do servidor, que tem de responder aos pedidos feitos pelo cliente;
- **Base de dados** : Toda a informação apresentada ao cliente deve estar guardada numa base de dados, com o intuito de ajudar a aplicação a responder aos pedidos do cliente;

Nos seguintes tópicos serão abordadas algumas linguagens pertencentes a cada tipo, explicando o funcionamento e a importância de cada uma delas, de modo a possibilitar um enquadramento com as tecnologias.

2.4.7.1 *HTML*

O HTML é uma linguagem de marcação que é utilizada para conceber a estrutura das páginas *web* e que utiliza *tags* para definir a estrutura da interface. O HTML tem uma estrutura base que é composta pelo *head*, sendo aqui que constam as informações, como por exemplo, o título da página, bem como, os ficheiros com extensões de CSS e *JavaScript*. Nesta linguagem existe o *body* onde é introduzido todo o conteúdo da página *web* e por fim temos o *footer* que é utilizado muitas vezes para colocar os direitos do *Website*, o mapa do *Website*, inclusão de ficheiros *JavaScript* entre outras opções. A estrutura é regulamentada pelas normas da *World Wide Web Consortium (W3C)*, ou seja, a instituição que rege as normas do desenvolvimento e interpretação de todo o conteúdo existente na *World Wide Web (WWW)*.

2.4.7.2 *CSS*

O CSS não é uma linguagem de programação, mas não deixa de ter um papel proeminente para que o *Website* ganhe forma e estilo, permitindo ao programador colocar o *Website* da forma pretendida. O CSS possibilita ainda ao programador isolar o código aplicado nas diversas páginas HTML, colocando tudo em um ou em vários ficheiros CSS e oferecendo assim uma manutenção e criação de código mais eficaz e rápida.

Para que sejam aplicados os estilos definidos nos ficheiros CSS é necessário associar os elementos HTML com classe e *id's* que constam dos ficheiros CSS com os seus elementos/estilos já definidos, oferecendo ao utilizador um *design* mais agradável. Esta tecnologia ainda tira partido das *media querys*, nomeadamente quando estamos perante dispositivos com várias resoluções, e as *media querys* permitem aplicar determinados estilos consoante a resolução do dispositivo.

2.4.7.3 *Bootstrap*

O *bootstrap*[4] é a *framework* mais conhecida para desenvolvimento *web* para as tecnologias HTML, CSS e *JavaScript*.

4 Página web: `http://getbootstrap.com/`

Esta tecnologia teve uma grande relevância para o desenvolvimento *web*, pois tornou possível que os *Websites* se tornam-se responsivos, isto é, ajustáveis à resolução do ecrã do utilizador. Para tal efeito o *bootstrap* tem implementado o sistema de linhas e colunas, onde cada número de colunas tem uma percentagem associada, para que se adapte ao dispositivo do utilizador. O número de linhas pode ser ilimitado, sendo que cada linha pode ter no máximo até 12 colunas.

No que respeita a exemplos onde se possa utilizar o *Bootstrap* em *JavaScript*, temos o exemplo do *modal*, que pode ser considerado como uma sub-página *web*, que possibilita ao utilizador visualizar informação sem ter que ir para outra página, permitindo assim uma usabilidade mais amigável para os utilizadores.

2.4.7.4 *JavaScript*

O *JavaScript*[5] é utilizado correntemente devido ao facto de ser uma linguagem que trabalha no lado do cliente e que possibilita que o *Website* tenha um melhor desempenho. Não existe a necessidade de uma ligação recorrente ao servidor e consequentemente possibilitando que a mesma consiga manipular os elementos HTML que têm classes e *id* associadas através do CSS, de modo a tirar partido do *Document Object Model (DOM)*. O DOM permite que o *JavaScript* consiga tirar partido dos eventos, propriedades dos métodos associados aos elementos da página *web*.

Esta linguagem foi muito importante para os programadores *web* porque possibilitaram aos mesmos melhorar a fluidez e interacção dos utilizadores com os *Websites*.

2.4.7.5 *AJAX*

O AJAX permite que se faça um pedido ao servidor da informação de que precisa, sem ser necessário carregar a página na totalidade. Esta tecnologia faculta que o *Website* se torne muito mais fluente e dinâmico, permitindo que as soluções *web* resolvam um problema que as afetava em relação às soluções de *software* nativas.

A título exemplificativo, e para provar a utilidade desta tecnologia, existindo a necessidade de selecionar um determinado país, o AJAX realizaria o pedido ao servidor que lhe responderia com a respectiva lista das cidades integrantes desse mesmo país, não sendo necessário recarregar a página *web* para esse efeito. Assim, há uma significativa melhoria da *performance*.

2.4.7.6 *jQuery*

O *jQuery*[6] é uma biblioteca de código *JavaScript* que corre no lado do cliente, esta tecnologia tem como grande objectivo tornar o desenvolvimento *web* mais fácil e permite ainda

5 Página web: `https://www.javascript.com/`
6 Página web: `http://jquery.com/`

programar *JavaScript* de uma forma mais fácil. Esta tecnologia é muito utilizada pelos programadores que incidem nas funcionalidades que interagem com os utilizadores, ou seja, efeitos de imagens, desenvolver animações aos elementos da página *web*, comunicação com a tecnologia AJAX, entre outras operações.

Com o surgimento de dispositivos móveis foi necessário incorporar no *jQuery* a biblioteca de *jQuery Mobile* onde tem funções definidas especificamente para esses dispositivo. O *jQuery* ainda tem bibliotecas como *jQuery UI* que se referem à biblioteca responsável pelos elementos da interface do utilizador e tem a biblioteca *Qunit* que permite realizar testes *JavaScript*.

2.4.7.7 *PHP*

O PHP [7] é utilizado principalmente para aceder a toda a informação que está presente na base de dados, sendo uma linguagem que trabalha do lado do servidor. Toda a informação que é apresentada no *Website* é através do PHP, onde é criado um ficheiro para aceder à base de dados do projecto e a gerir toda a informação.

O PHP faz a ligação na arquitectura de *software* MVC entre o *controller*, *model* e a *view*, possibilitando que as aplicações *web* se tornem dinâmicas. O *controller* é responsável por responder ao pedido efetuado pelo utilizador através da *view*, no entanto caso o *controller* precise de informação da base de dados, deve consultar o *model*, de modo a oferecer a informação solicitada pelo utilizador. O PHP é uma linguagem muito utilizada, sendo das mais utilizadas em soluções *web*[9], tem a particularidade de suportar OOP, levando à utilização de classes e à redução do nível de SQL *injection*.

O PHP com a última versão implementada, aumentou significativamente a *performance* desta linguagem, tirando partido da contribuição da *ZEND*[8] e do projecto *HipHop Virtual Machine (HHVM)*[9]. O ZEND é uma companhia responsável pela revolução que o PHP sofreu. O HHVM foi um projecto que surgiu com o aparecimento do *facebook*, de modo a garantir uma interpretação do código em *real-time*, ou seja, mais rápido que a versão 5.6 do PHP.

2.4.7.8 *ASP.NET*

O *ASP.NET*[10] é uma tecnologia que é utilizada para aplicações *web* e para isso tira partido da *framework .NET*, esta tecnologia não é *opensource* e a sua criadora é a *Microsoft*. A mesma pode ser utilizada por *C#*, *C++* e *Visual Basic*. Esta *framework* tem diversas bibliotecas que ajudam na concepção de um código e funcionalidades eficazes.

Para o seu funcionamento esta tecnologia tira partido de dois módulos importantes:

7 Página web: `https://secure.php.net/manual/en/index.php`
8 Página web: `http://www.zend.com`
9 Página web: `http://hhvm.com/`
10 Página web: `https://www.asp.net/`

- ***Commom Language Runtime (CLR)***: Este módulo é responsável por converter o código fonte, de modo a ser executado por uma máquina virtual. Este módulo é responsável pela compatibilidade de várias linguagens utilizadas nesta tecnologia. Esta funcionalidade é um dos pontos fortes desta tecnologia.

- ***Framework Class Library (FCL)***: É responsável pela utilização das principais bibliotecas utilizadas para desenvolver aplicações com esta tecnologia.

O *ASP.NET* utiliza como a *design pattern* MVC, de modo a estruturar o código da aplicação, ou seja, cada vez mais se têm definido um *design pattern* para os produtos *web*, com o intuito de os programadores se adaptarem com facilidade.

2.4.8 *Base de dados*

2.4.8.1 *MySQL*

O *MySQL*[11] é dos *Sistema de Gestão de Base de Dados (SGBD)* mais utilizados no desenvolvimento *web* a nível mundial[10], pois tem a vantagem de requerer pouco hardware e tem a particularidade de ser muito rápido a desenvolver todas as transações executadas no SGBD. Este SGBD tira partido da compatibilidade com vários *storage engines*, nomeadamente *InnoDB*, *MyISAM*, entre outros. Permite ainda utilizar procedimentos armazenados, *triggers* e cursores conseguido-se equiparar com as funcionalidades oferecidas pelos seus concorrentes.

Esta tecnologia permite ainda a compatibilidade com diversas linguagens que são executadas no lado do servidor como PHP, *ASP.NET* entre outras. A figura 12 ilustra a arquitectura utilizada pelo *MySQL*.

11 Página web: `https://www.mysql.com/`

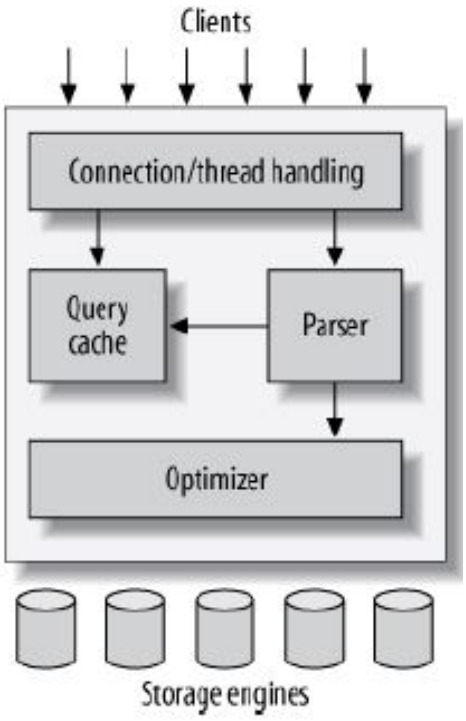


Figura 12.: Arquitectura do *MySQL*
[11]

Na figura 12 é possível verificar os elementos da arquitectura do *MySQL*. A primeira camada é responsável pela autenticação ao servidor *MySQL*, bem como, à base de dados em questão. Na segunda camada temos presente o *Query cache*, *Parser* e *Optimizer*. Esta camada é responsável pelas *querys* realizadas às base de dados, a funções ligadas aos dados, criptografia, procedimentos armazenados, *triggers* entre outras operações, tirando partido da *cache* referente às *querys* realizadas .

Por fim, na terceira camada, estão presentes os *store engines*, que são responsáveis por guardar toda a informação. O servidor *MySQL* comunica com os *store engines* através de uma API, onde a mesma tem funções de baixo nível que executam o início de transações, ou mesmo o início de uma consulta[11]. Os *store engines* mais utilizados é o *InnoDB* e *MyISAM*, no capítulo 4 iremos abordar com mais detalhes cada um destes *store engines*.

2.4.9 *Framework*

Uma *framework* é uma abstração na qual o *software*, que produz funcionalidades genéricas, pode ser mudado com código adicional feito pelo utilizador da mesma. Permite-se juntar códigos de diversos projectos, nomeadamente numa *framework* criada pelo programador, aumentando a generalidade do programa, bem como, o seu rápido desenvolvimento[12].

Uma *framework* é uma colecção de funções e métodos que já foram testados/implementados e que podem ser utilizadas a qualquer momento.

A utilização de *frameworks* permite que se reutilize código que já tenha sido testado, possibilitando ainda tirar partido de uma manutenção mais eficiente visto ter que corrigir o código apenas uma vez. Quem desenvolver uma *framework* tem que produzir código o mais genérico, com vista a sua utilização em diversos projectos.

No que diz respeito às *frameworks* relacionadas com o desenvolvimento *web* ,tendem a seguir o padrão MVC, dividindo a aplicação *web* em três partes interconectadas. Esta divisão torna clara a separação entre os objectos de domínio e da interface da visualização do utilizador, como arquitectura de *software*. É necessário realçar que utilizam OOP, tal como a esmagadora maioria dos *frameworks*, o que resulta em que as classes *models* são, regra geral, diretamente mapeadas em tabelas nas base de dados, sem qualquer intervenção direta do programador. No *controller* o programador vai criar um objecto associado às tabelas que pretende trabalhar, facilitando-lhe assim toda a interacção com as tabelas da base de dados. É ainda necessário salientar que a nível *web* estas *frameworks* já tem funções definidas para as consultas, o que torna mais simples quer a recolha de dados quer o aumento do desempenho do *Website*/aplicação *web*.

2.4.10 *OOP*

O paradigma de OOP surgiu para oferecer aos programadores uma nova forma de estruturar a informação e de como aceder à mesma nos seus programas. No diagrama de classes já foram abordados alguns aspectos sobre as classes, nomeadamente a visibilidade (*Private*, *Public*, *Protected* ou *Package*) e o tipo de associações. Nesta secção 2.4.10 vamos nos focar mais em aspectos relacionados com o código.

A classe é responsável por definir a estrutura de um objecto, ou seja, a classe define os atributos e métodos que o objecto dispõe. O objecto é uma instância de uma classe, sendo que cada objecto tem os seus valores o que proporciona uma diferença entre os outros objectos da mesma classe.

```
// Classe Player
class Player {

// Primeiro sao definidos os atributos
public $playerName;
public $playerPosition;

// Construtor do objecto
public function __construct(string $name, string $position) {
  $this->playerName     = $name;
  $this->playerPosition = $position;
```

```
  }

//Seguidamente define-se os metodos
 public function getName(){
    return $this->playerName;
 }

 public function setName(string $name){
    $this->playerName = $name;
 }

 public function getPosition(){
    return $this->playerPosition;
 }

 public function setPosition(string $position){
    $this->playerPosition = $position;
 }
```

Listing 2.1: Exemplo de uma classe

```
  // Objecto da classe Palyer
$player = new Player("Artur Jorge", "Defesa");

$player->getName(); // Vai ser retonado o nome do jogador Artur Jorge
$player->setName("Ricardo"); // Vai atribuir um novo nome ao objecto
echo $player->name; // Vai ser retonado o nome do jogador Ricardo
```

Listing 2.2: Instanciar uma classe

Este código representa a estrutura da classe *Player*. Foram definidos os seus atributos e métodos, sendo que os métodos foram apenas os *getters* e *setters* que podem ser considerados como os métodos base de cada classe. No seguinte código é possível verificar uma instância de um objecto, ou seja, é feita a construção de um objecto que passa a ter a estrutura da classe pertencente. Posteriormente serão utilizados alguns métodos da classe, onde obtemos o nome do jogador e seguidamente alteramos o nome do jogador.

Este paradigma também tira partido de outras características relacionadas com as classes, tais como:

- ***Encapsulation***: *Encapsulation* pode ser considerado como um mecanismo de segurança, devido à definição de quais são os métodos e atributos disponíveis para outros objectos. O seu uso torna-se importante, para tornar o *software* mais flexível a novas alterações;

- ***Inheritance***: Esta propriedade permite às classes filhas que herdem todos os métodos e atributos da classe Pai, reduzindo o código a implementar e a definir as suas diferenças para as outras classes filhas. O seu uso é aconselhado quando nos deparamos com classes muitos semelhantes. Por exemplo no diagrama de classes apresentado temos a classe Conta como classe Pai e depois as suas filhas apenas têm métodos que as distinguem umas das outras.

- ***Abstract***: As classes abstratas são as classes que não podem ser instanciadas. Esta propriedade é muito utilizada em herança, quando pretendemos que as classes filhas implementem determinados métodos. Com esta propriedade é obrigatório que as classes filhas implementem os métodos da classe pai que são abstractos.

- ***Polymorphism***: Esta característica permite que várias classes tenham métodos com o mesmo nome, mas com comportamentos diferentes. O uso de polimorfismo, só faz sentido ser utilizado quando estamos perante uma herança de classes. O polimorfismo permite reduzir o código e torna a aplicação mais flexível.

- ***Interface***: A *interface* é responsável por agregar diversos métodos só que apenas têm a sua assinatura, ou seja, os métodos não podem conter qualquer implementação. Não é possível instanciar uma interface, pois não é possível criar um construtor. A vantagem desta característica do OOP é definir os métodos que as classes que implementem esta interface devem seguir.

2.4.11 *API*

A API em desenvolvimento *web* é algo que permite que determinadas aplicações consigam comunicar entre si. Quando o programador desenvolve a API, vai definir *endpoints*, com determinada funcionalidade. Assim a aplicação faz um pedido a API através desse *endpoint*, de modo a obter a resposta.

O recurso a esta tecnologia é muito usual quando se desenvolve uma aplicação *mobile*. Depois de ter uma aplicação *web*, a aplicação *mobile* não vai necessitar de ter acesso a uma base de dados, necessitando apenas de chamar os *endpoints* da API e apresentar a informação requerida pelo utilizador. Contudo esta solução também é muito utilizada quando se pretende fazer comunicações entre plataformas e assim qualquer plataforma que esteja autorizada pode utilizar determinada API. É necessário salientar que as API por vezes utilizam API-*Keys* para garantir que quem fez a solicitação tem permissões para o efeito.

Existem dois tipos de API, que vão ser descritas com mais detalhe nos tópicos seguintes.

2.4.11.1 Representational State Transfer (REST)

O REST é um tipo de API que opera sobre o protocolo HTTP, de modo a interpretar o pedido das aplicações que recorrem aos seus *endpoints*. A sua comunicação é sempre realizada sobre o protocolo HTTP, sendo que a grande diferença para o servidor HTTP é que a API apenas devolve uma resposta em *JavaScript Object Notation (JSON)* ou *eXtensible Markup Language (XML)*.

Esta API utiliza muitos dos códigos do protocolo de HTTP, de forma a identificar o estado da comunicação entre os serviços. Seguidamente vão ser apresentadas a gama de códigos HTTP utilizado pelo REST:

- **1XX: Informativo**: Esta gama de códigos refere-se a uma resposta temporária, apenas contendo o cabeçalho associado ao pedido;
- **2XX: Sucesso**: Quando recebemos um código desta gama, quer dizer que o pedido foi tratado com sucesso;
- **3XX:Redireccionamento**: No que respeita aos códigos desta gama, especifica que o cliente precisa de completar outra acção para utilizar aquele *endpoint*. Dai esta gama ser considerada a gama do redireccionamento;
- **4XX Erro do cliente**: Estes códigos representam um erro ao nível do cliente, ou seja, do emissor do pedido. Por norma este erro acontece quando o cliente realizou mal o pedido, o *url* não existe ou até mesmo quando não tem autorização para o pedido;
- **5XX Erro no servidor**: Quando a API devolve um código desta gama, quer dizer que ocorreu um erro no servidor. O mais usual é quando existe um erro a consultar a base de dados ou mesmo a nível da aplicação que trata o pedido, *timeout* por exemplo.

O REST também divide os seus *requests* em métodos que também são denominados por verbos. Cada método tem uma finalidade e funcionamento diferente dos outros. Seguidamente vão ser apresentados os mais utilizados:

- ***GET***: Quando o *endpoint* utiliza o método *GET*, quer dizer que o objectivo é devolver informação sobre determinado recurso. Por exemplo informação de um jogo, ou de um jogador;
- ***POST***: O *POST* é utilizado para inserir um registo na base de dados, ou seja, permite que através de um *endpoint* se consiga inserir vários jogadores, sendo que os dados são enviados no pedido;
- **PUT**: O *PUT* tem o mesmo objectivo que o *POST*, a única diferença é que este método pode apenas, por exemplo, actualizar um jogador enquanto que o *POST* pode inserir informação em vários jogos;

- **DELETE**: O *DELETE* é utilizado quando pretendemos remover algum recurso, deve ser utilizado quando é pretendido remover um jogador da base de dados;

2.4.11.2 Simple Object Access Protocol (SOAP)

O SOAP é um protocolo aplicado para que seja possível trocar informação entre diversas aplicações. As mensagens deste protocolo utilizam o XML.

As mensagens de SOAP são divididas nas seguintes partes:

- **Envelope**: Esta parte representa o início e o fim da mensagem que será enviada para a outra plataforma;
- ***Headers***: Deve especificar informações referentes à aplicação, que não são especificadas no corpo da mensagem, ou seja, elementos extras, como autenticação entre outros;
- ***Body***: O *body* é a parte que contem a informação a passar a outra plataforma. É nela que é colocada a informação que a outra plataforma precisa, para tratar do pedido realizado;
- **Fault**: Responsável por identificar os erros que ocorrem durante o transporte das mensagens. Os erros são identificados por sub-elementos, de modo a despistar o erro ocorrido na comunicação;

O transporte das mensagens SOAP pode ser realizado por diversos protocolos desde *File Transfer Protocol (FTP)* a *Simple Mail Transfer Protocol (SMTP)*, sendo que o mais utilizado é o HTTP. O transporte das mensagens é enviado pelos métodos HTTP, nomeadamente o *GET* e *POST*.

Este protocolo é muito conhecido pelo seu rigor nas mensagens, devido à estrutura que é definida para cada mensagem, uma vez que basta não ter um elemento definido na mensagem para que haja falha na mensagem a enviar.

2.4.12 *Segurança*

Este ponto tem sido alvo de um estudo exaustivo por toda a comunidade *web*, uma vez que os utilizadores de todas as soluções/produtos *web* pretendem navegar e usufruir das tecnologias *web* com garantia de segurança, integridade dos seus dados pessoais como das credenciais de acesso.

A *Open Web Application Security Project (OWASP)*[13] é uma fundação que ajuda a comunidade *web* a conceber aplicações seguras para os seus utilizadores, cabendo-lhe alertar para vários ataques que existem no mundo das aplicações *web*.

Seguidamente vão ser abordados alguns desses ataques e as suas prevenções.

2.4.12.1 *SQL injection*

O SQL injection é um ataque que é responsável por manipular consultas que a aplicação *web* faça à base de dados. Para que o ataque tenha sucesso basta ao intruso introduzir código SQL na *querystring* ou no *input* (elemento do HTML), permite àquele explorar a vulnaberidade a aplicação, acedendo a base de dados, para alterar os valores ou nos casos mais extremos, poder apagar toda a base de dados.

De acordo com OWASP existem vários pontos a ter em atenção para evitar ataques de SQL *injection*, tais como [14]:

- Evitar fazer *querys* à base de dados com dados introduzidos diretamente pelo utilizador. Deve ser feita sempre uma parametrização das consultas de modo a evitar o SQL *injection*;
- Utilizar procedimentos armazenados ou funções;
- Utilizar funções de *"escape"* de todos os dados introduzidos pelos utilizadores;
- Limitar privilégios aos acessos.

Nos seguintes exemplos é possível validar alguns exemplos de SQL *injection*. No primeiro caso é feito SQL *injection* pelos *inputs* do HTML. O utilizador depara-se com o formulário de *login* e vai colocar estes valores **"or ="** tanto no *username* como *password*. Quando o mesmo fizer *login* o sistema vai fazer a seguinte *query*.

```
SELECT * FROM Utilizadores WHERE username ="" or ""="" AND password ="
    " or ""=""
```

Listing 2.3: SQL *injection* exemplo de vulnerabilidade

Esta *query* vai possibilitar ao intruso verificar todos os registos presentes na tabela utilizadores o que consubstancia uma falha de segurança enorme. Pode também, por exemplo, colocar nos *inputs* ***DORP TABLE* Utilizadores** e com isto eliminar essa tabela.

Outra via de fazer o ataque é por *url*, ou seja, quando estamos perante o método *GET*, em que o intruso manipula os parâmetros. No *link* abaixo o utilizador manipulou o *id*.

```
http://www.apptese/utilizador/edit?id=1 or 1 = 1
```

Listing 2.4: SQL *injection* exemplo por pedido *GET*

Este processo vai obter o mesmo resultado que o ataque anterior. Os programadores devem seguir os conselhos da OWASP anteriormente mencionados, com intuito de evitar este tipo de ataques e garantir a segurança da sua base de dados, bem como, a dos seus utilizadores.

2.4.12.2 *CSRF*

Este ataque é muito usual quando o invasor pretende que os utilizadores da aplicação façam algum pedido indesejado à aplicação. Este ataque tem como principal objectivo alterar dados como *email* ou mesmo transferir fundos, devido ao invasor não obter resposta do servidor [15]. Por exemplo um *Website* que venda produtos, para que o atacante reproduza o ataque, tem que verificar que tipo de dados e qual o *url* necessário para fazer o ataque. Depois o atacante tem que introduzir no *Website* vulnerável uma imagem ou um *link*, de modo a fazer o redireccionamento para o seu site onde irá realizar o ataque. O utilizador ao clicar nesse elemento vai correr o *script* abaixo descriminado. O atacante consegue fazer uma compra sem que o próprio utilizador tenha pedido essa solicitação. O seguinte excerto de código mostra um possível ataque.

```
<!DOCTYPE HTML>
<html>
    <body>
        <form method="POST" action="http://servidor/csrfattack.php">
            <input type="hidden" name="produto_id" value="10">
            <input type="hidden" name="produto_preco" value="100">
            <input type="hidden" name="utilizador_id" value="1">
        </form>
        <script>
            document.forms[0].submit(); // Responsavel por fazer o
                envio do pedido
        </script>
    </body>
</html>
```

Listing 2.5: CSRF - vulnerabilidade

Posteriormente o servidor do atacante vai realizar o pedido ao *Website*, e com isso vai ser realizada a venda. No caso da aplicação ter sistema de *login*, o mesmo não irá levantar problemas porque o utilizador já está autenticado na aplicação, sendo que o atacante já sabe o *id* de sessão do utilizador por ter recebido um pedido por parte do mesmo.

Para evitar estes ataques, a solução é criar um *token* aleatório sempre que é feito um pedido ao servidor, ou seja, esta solução permite que o atacante não conheça todos os elementos para realizar o pedido. O utilizador acede à compra do produto e é gerado um *token* que é guardado na sessão e num *input* do pedido, de modo a verificar se o *token* é verdadeiro. Esta medida permite que o ataque de CSRF seja evitado.

2.4.12.3 *XSS*

(XSS) é um ataque que consiste no intruso colocar código em *JavaScript* em elementos HTML como *input* ou *textarea*. Um exemplo muito comum é realizar este ataque nas páginas *login*

para obter as credenciais dos utilizadores, sendo colocado um código *JavaScript* que simula as funcionalidades de *login*. O utilizador coloca as credenciais de forma normal só que ao fazer *submit* e como aquela funcionalidade está simulada no código de *JavaScript*, será enviado para outro servidor e com isso o intruso obtem as credenciais do utilizador.

Para evitar este ataque convêm que os programadores bloqueiem todo o código *JavaScript* e HTML nos elementos de entrada HTML.

Todos os programadores *web* devem proteger as suas aplicações contra estes tipos de ataque, pois é fundamental garantir fiabilidade aos seus utilizadores, pois qualquer utilizador, ao saber que certa aplicação não é segura, ou que já foi atacada pelo menos uma vez ,vai ficar apreensivo perante essa mesma aplicação.

2.5 TESTES

Depois de o produto estar implementado, devem ser realizados testes para validar se o produto foi bem implementado e se não contem erros, para que seja entregue ao cliente sem qualquer anomalia.

Esta fase é crucial para validar se os requisitos foram bem implementados, ou seja, verificar se a implementação cumpre os requisitos e os pressupostos para que aplicação tenha o funcionamento esperado.

Nos testes é fundamental avaliar todos os cenários positivos e negativos, de modo a validar a robustez do produto em questão.

2.6 DEPLOYMENT

Esta fase é responsável por disponibilizar o produto ao cliente. No que respeita ao *deployment* é importante configurar o ambiente onde o produto vai estar alojado. A equipa de desenvolvimento é responsável por identificar os serviços e dependências que a aplicação tem para funcionar corretamente.

Nesta fase o produto está pronto a ser utilizado pelos utilizadores e o produto entra numa fase de gestão, podendo o produto de *software* evoluir através da implementação de novas funcionalidades e melhorias, de forma a não estagnar.

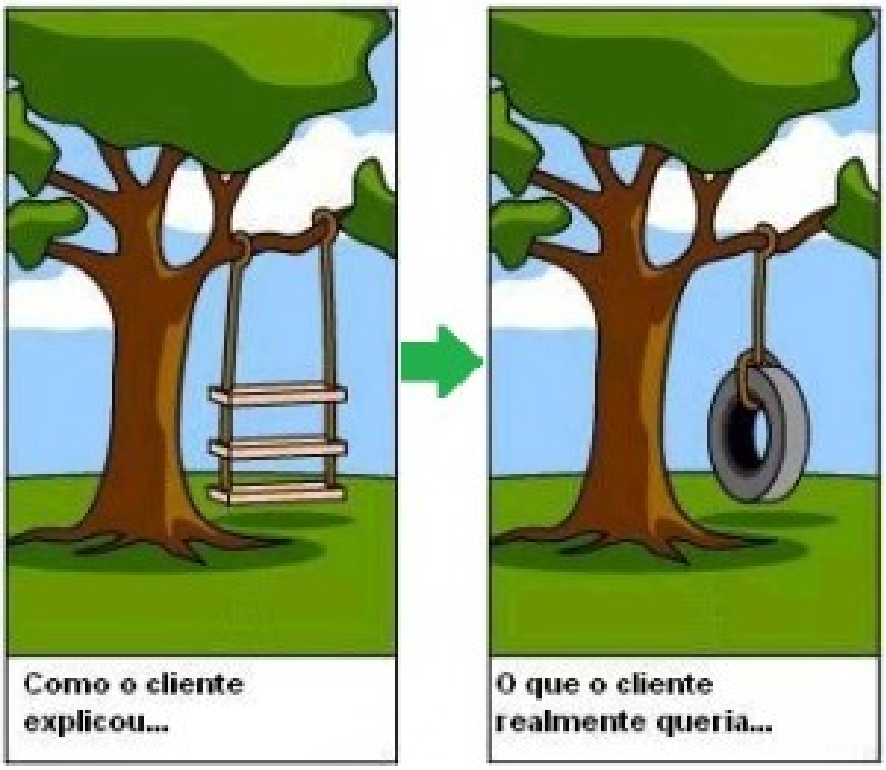


Figura 13.: O problema de um mau levantamento de requisitos

Em suma, no processo de *software*, é necessário seguir estes passos, de modo a conceber o produto com a melhor qualidade e celeridade, sendo o aconselhável ir recolhendo informações junto do cliente, para evitar que no último passo o cliente diga que não foi aquilo que pediu, como ilustra a figura 13.

3

ANÁLISE E CONCEPÇÃO

Neste capítulo vai ser detalhado parte do processo de desenvolvimento de um sistema de *software*. Em causa estará o processo desenvolvimento da aplicação, relacionada com a história de um clube de futebol. Este processo vai passar pelas seguintes fases:

- **Análise**: Nesta etapa serão apresentadas as técnicas utilizadas para obter os requisitos;
- **Concepção**: A concepção vai ser responsável por apresentar os diagramas concebidos, de modo a definir um *workflow* para determinado requisito.

Esta dissertação tem como objectivo dar resposta à necessidade que os adeptos de futebol têm de consultar toda a informação referente à história do seu clube. Pretende-se que os utilizadores também consigam consultar as estatísticas de um determinado jogador do seu clube com determinada equipa adversária, sendo que também o podem fazer em relação a equipas adversárias, de modo a verificar, por exemplo, se a sua equipa é favorita para um determinado jogo.

3.1 ANÁLISE

No que diz respeito à análise do problema em questão, foi necessário começar por entender quais as funcionalidades mais importantes a integrar, de modo a garantir a resolução dos problemas, bem como oferecer uma interacção agradável com a aplicação *web*. Para descobrir quais as funcionalidades necessárias a implementar foi necessário conceber uma primeira versão do documento de requisitos.

Foram levantados requisitos funcionais que se referem às funcionalidades da aplicação *web*, bem como requisitos não funcionais referentes ao comportamento da mesma. Um dos requisitos analisados foi ser ou não responsivo, uma vez que cada vez mais se utilizam aplicativos móveis, sendo por isso fulcral adaptar a aplicação a esses dispositivos.

3.1.1 *Técnicas utilizadas*

Para obter os requisitos de qualquer aplicação é necessário utilizar algumas técnicas. Serão apresentadas as abordagens utilizadas para o levantamento de requisitos desta aplicação. Como foi demonstrado no capítulo 2, este passo é fundamental para definir e perceber o que o cliente pretende.

3.1.1.1 *Análise do domínio*

A análise do domínio é importante para estudar soluções já existentes, com intuito de conceber um produto que consiga colmatar as falhas de implementação, que se achem necessárias de integrar, de modo a proporcionar curiosidade ao seu público alvo. É necessário realçar que foi pedida uma colaboração com a plataforma zerozero, mas sem nunca se obter uma resposta por parte daquela entidade.

O primeiro passo foi estudar a área em que se incide o projecto, analisar o tipo de soluções já existentes, ou seja, ver como guarda a informação, entre outras. O estudo incidiu nas seguintes soluções:

- **ZeroZero:** O zerozero[1] é das plataformas mais conhecidas em Portugal. Este *website* fornece dados de várias ligas e competições. Contem informações de diversos jogadores e informação de clubes profissionais e amadores. É necessário salientar que a informação dos clubes amadores está menos aprofundada, o que pretendemos colmatar com a implementação da aplicação a desenvolver. Um ponto forte do zerozero é a análise de confrontos entre equipas e jogadores que apresentam, algo que qualquer amante do desporto gosta de visualizar.

- ***Websites* de clubes:** A segunda análise foi de diversos *websites* de clubes, análise essa que não ficou limitada à área do futebol, tendo sido analisados *websites* de outros desportos. No estudo feito sobre os *websites* das equipas portuguesas, foi possível verificar que maior parte das equipas, apenas guarda os seus resultados, utilizando até soluções de outros, mas nenhum deles fornece estatísticas sobre os jogadores ou mesmo confrontos entre equipas e jogadores. Este foi um dos pontos que se achou fundamental implementar, de forma a diferenciar o nosso produto dos outros já existentes.

Em suma, o zerozero é uma plataforma com muita informação e bem detalhada, mas no entanto foi necessário comparar soluções com o mesmo objectivo da aplicação a desenvolver. Esta análise ajudou a potencializar a nossa aplicação, o que originou os requisitos de confrontos e estatísticas dos jogadores e da equipa.

1 Página Web: http://www.zerozero.pt

3.1.1.2 *Entrevistas*

As entrevistas neste caso, foram mais informais, nunca sendo definido um guião de perguntas para obter os requisitos. Foram realizadas algumas reuniões com o cliente, sendo necessário evidenciar que o cliente já tinha ideias organizadas, facilitando assim a obtenção dos respectivos requisitos.

Esta técnica é fundamental para perceber quais as ideias do cliente, cabendo ao analista realizar certas perguntas, com intuito de reforçar as ideias que o cliente pretende, surgindo algumas alterações às suas ideias iniciais.

No caso concreto desta análise, as entrevistas podem ser consideradas reuniões, dada a forma como foram conduzidas e como se foram obtendo os requisitos.

3.1.1.3 *Introspeção*

A introspeção é uma técnica bastante utilizada quando o analista ou engenheiro de *software* se encontra dentro do domínio da aplicação. Permite que o mesmo identifique possíveis valias para a aplicação a desenvolver, sendo necessário referir que esta técnica é baseada na análise de domínio, o que permite verificar as fraquezas e outras soluções a implementar.

O analista ao utilizar esta técnica vai vestir a pele de um utilizador do produto e com isso vai enumerar as funcionalidades que gostaria de ter na aplicação caso fosse ele um utilizador.

3.1.2 *Levantamento dos requisitos*

No que respeita ao levantamento de requisitos, é altura de definir as funcionalidades que a aplicação irá conter, ou seja, vão ser escritos os requisitos com base nas técnicas anteriormente mencionadas.

3.1.2.1 *Cartão de volere*

Para a escrita de requisitos foi utilizado o cartão *volere*, como foi explicado no capítulo 2, este cartão é um *template* utilizado pelos analistas, que fundamenta não só o requisitos, bem como, a sua fundamentação ou o seu critério de ajuste.

Seguidamente (Fig.14) é possível analisar um exemplo da escrita de um dos muitos requisitos da aplicação.

Requisito:	28	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consulta confrontos de jogadores
Descrição:	O utilizador consulta estatísticas de confrontos de jogadores com equipas adversárias				
Fundamentação:	Possibilita ao utilizador consultar as estatísticas referentes a um jogador contra uma determinada equipa numa determinada competição				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador seleciona um jogador e posteriormente visualiza as estáticas do confronto que escolheu				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito				

Figura 14.: Cartão de volere referente a um requisito

No cartão de *volere* é possível verificar toda a informação associada ao requisito, desde o seu tipo a sua fundamentação, bem como, o seu critério de ajuste, de modo a permitir que quando for realizado o teste se saiba o resultado a obter. O requisito irá será definido no campo da descrição, contudo os requisitos devem ser claros e simples, de modo a evitar as ambiguidades.

3.1.2.2 *Levantamento de requisitos*

Os requisitos levantados para aplicação podem ser consultados com mais detalhe nos anexos A. Seguidamente apresentam-se todos os requisitos funcionais que foram identificados:

1. O administrador insere/edita/consulta/elimina detalhes de um jogador;
2. O administrador procura um jogador;
3. O administrador insere/edita/consulta/elimina informação sobre os jogos associados à equipa;
4. O administrador pesquisa jogos do clube;
5. O administrador insere/edita/lista/elimina os eventos associados ao jogo;

6. O administrador pesquisa eventos;
7. O administrador insere/edita/consulta todos as estatísticas relativas ao jogo;
8. O administrador regista/altera/consulta/remove informação de um árbitro;
9. O administrador pesquisa árbitros;
10. O administrador regista/altera/consulta/remove informação de um estádio;
11. O administrador procura estádios;
12. O administrador insere/altera/lista/elimina as equipas adversárias;
13. O administrador pesquisas equipas adversárias;
14. O administrador regista/altera/consulta/elimina informação do treinador;
15. O administrador pesquisa treinadores;
16. O administrador insere/edita/elimina as competições que a equipa disputa;
17. O administrador procura as competições;
18. O administrador insere/altera/consulta/remove informação referente a época desportiva;
19. O administrador pesquisa época desportiva;
20. O administrador cria/edita/consulta/elimina notícias;
21. O administrador procura notícias;
22. O administrador insere/edita/consulta/elimina eventos históricos associado a equipa;
23. O administrador consulta eventos históricos associados a equipa;
24. O administrador insere/edita os dados pessoais;
25. O administrador autentica-se na aplicação;
26. O utilizador consulta informações sobre determinado jogo;
27. O utilizador consulta estatísticas sobre os jogadores;
28. O utilizador consulta estatísticas de confrontos de jogadores com equipas adversárias;
29. O utilizador consulta estatísticas de confrontos com equipas adversárias;
30. O utilizador insere/edita os dados pessoais;

31. O utilizador visualiza dados sobre jogadores em determinada época;
32. O utilizador consulta o desempenho da equipa em determinada competição;
33. O utilizador regista-se na aplicação;
34. O utilizador autentica-se na aplicação;
35. O utilizador utiliza a conta do *facebook* para se autenticar;
36. O utilizador recupera a *password*;
37. O utilizador consulta notícias;
38. O utilizador consulta histórias;
39. O sistema valida o *login*;

    No que respeita aos requisitos não funcionais, que são responsáveis por definir os atributos ou qualidades do produto, temos a seguinte lista:

40. A interface da aplicação *web* adapta-se a qualquer dispositivo;
41. A aplicação *web* oferece usabilidade aos utilizadores;
42. A aplicação *web* deverá garantir que não existem acessos não autorizados ao *backend*;
43. A aplicação *web* garante escalabilidade;
44. A aplicação *web* oferece um tempo de resposta curto aos utilizadores.

3.2 CONCEPÇÃO

A concepção é responsável por transcrever todos os requisitos em diagramas, de modo a permitir que o programador tenha a vida facilitada quando começa o desenvolvimento do produto. No entanto, é necessário realçar que apenas vão ser apresentados alguns casos de diagramas de casos de uso, sequência e actividades, uma vez que, para cada requisito, foi associado cada um dos diagramas elencados.

3.2.1 *Modelo de domínio*

Na opinião dos autores João Miguel Fernandes e Ricardo Machado "Um modelo de domínio constitui uma descrição das propriedades comuns e variáveis do domínio relacionado com o sistema de software que está a ser desenvolvido. O modelo de domínio

expressa verdades duradouras sobre o universo que é relevante ao sistema em desenvolvimento.

Essa descrição deve incluir (1) a definição do âmbito desse domínio, dando exemplos de sistemas nele incluídos ou regras genéricas de inclusão, (2) o vocabulário do domínio (i.e., o glossário com os principais termos), e (3) um modelo de conceitos que identifica e relaciona os conceitos desse domínio. "(Tradução nossa)[2].

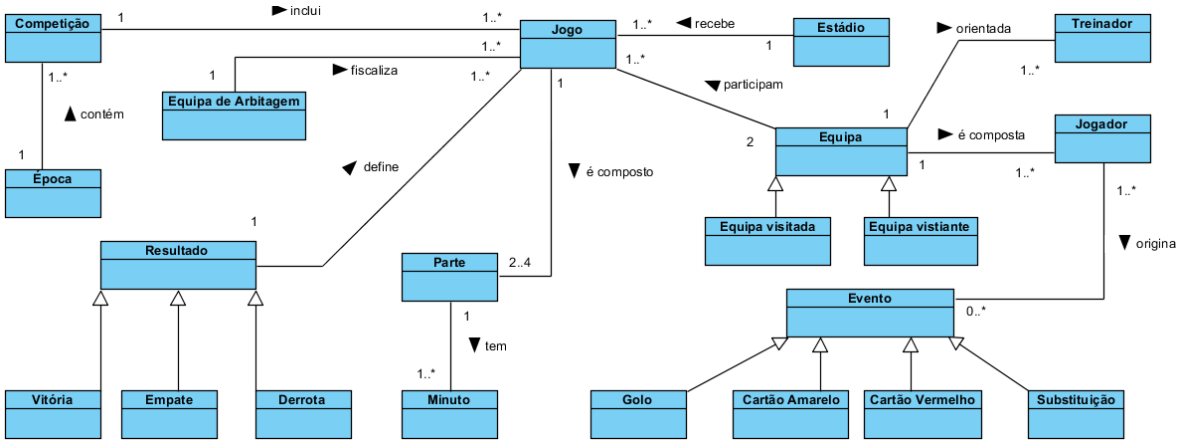


Figura 15.: Modelo de domínio da aplicação

Na figura 15 é possível verificar o modelo de domínio concebido para este produto. Com este diagrama é possível evidenciar certas características associadas ao modelo em questão, tais como:

- Uma época contém uma ou mais competições, que por sua vez têm diversos jogos associados;
- Um jogo em geral é divido em duas partes, no caso de existir prolongamento pode ter quatro partes, compostas por vários minutos, no entanto um jogo apenas pode ter um de três resultados possíveis (vitória, empate ou derrota);
- Um encontro é realizado num estádio, que por sua vez é disputado por duas equipas e fiscalizado por uma equipa de arbitragem;
- Uma equipa ao longo de uma época pode ser orientada por um ou mais treinadores, que escolhe os seus jogadores para cada jogo;
- No que respeita ao jogo em si, ele pode ter diversos eventos, constituindo a história do jogo, permitindo calcular certas estatísticas. Com isto, é possível analisar cada equipa e cada jogador em diversos jogos.

Em suma, este diagrama é bastante importante, uma vez que permite ao analista enquadrar as necessidades do domínio em causa, definindo assim uma visão mais abrangente a todos os envolvidos no projecto.

3.2.2 *Diagrama de classes*

O diagrama de classes define os métodos e atributos associados a cada classe. Este diagrama vai possibilitar ao programador estruturar o seu código e as funções que tem que implementar, bem como, a estrutura do projecto.

Esta abordagem é uma boa base para o desenvolvimento do modelo ER, facilitando a construção do mesmo, visto que já tem representado a relação entre classes (que representam as entidades do modelo ER). No entanto, é preciso referenciar que os atributos de cada classe serão os campos de determinada tabela.

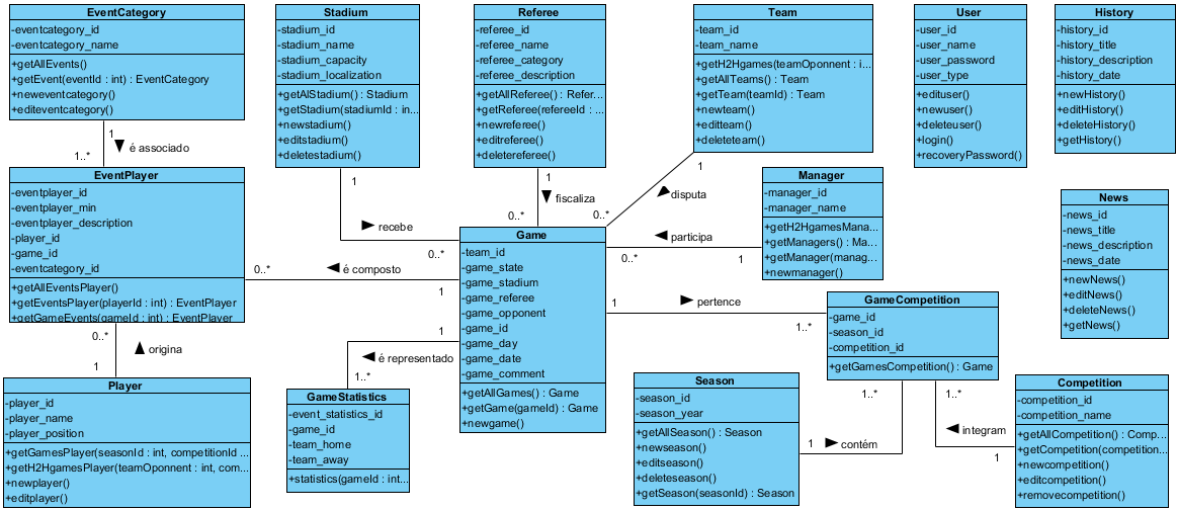


Figura 16.: Diagrama de classes

Na figura 16 é possível verificar o diagrama de classes referente à aplicação, sendo necessário salientar que não é possível apresentar todos os métodos que cada classe tem. Este diagrama permite esclarecer as seguintes características:

- A classe ***Game*** pode ser considerada como o *core* da aplicação, ou seja, a maior parte das classes estão ligadas à mesma. Por exemplo, esta classe vai ter definidos os métodos para apresentar a informação de cada jogo, como a lista de todos os jogos;
- A classe ***EventPlayer*** é responsável por manipular a informação sobre todos os eventos de um determinado jogador num ou mais jogo;
- A classe ***GameStatistics*** é responsável por controlar todas as estatísticas relacionadas com o jogo e respectiva equipa;

- As classes ***Stadium***, ***Referee*** e ***Team*** permitem manusear a informação sobre a respectiva classe, complementando a informação dos jogos. A classe ***Team*** tem um método para apresentar os confrontos entre determinadas equipas;
- A classe ***Competition*** é responsável por tratar dos jogos que estão associados a determinada competição;
- A classe ***GameCompetition*** é essencial para associar a informação do jogo, mais concretamente a que época e a que competição pertence. Esta classe vai facilitar o cálculo das estatísticas pois permite descobrir em que jogos determinado jogador/treinador participaram;
- A classe ***Manager*** é responsável por tratar da informação referente ao treinador, sendo responsável por apresentar os jogos em que determinado treinador participou;
- A classe ***Player*** é responsável por apresentar os dados referentes ao jogador e aos seus confrontos com determinadas equipas.

3.2.3 *Modelo ER*

O modelo ER tem como objectivo apresentar e definir de que forma a informação vai ser armazenada, de modo a garantir a consistência dos dados.

A base de dados vai ter o modelo ER abaixo representado (Fig.17), onde possivelmente existirão algumas mudanças consoante novas funcionalidades que possam surgir na aplicação *web*. A figura 17 representa o modelo ER da aplicação *web* em questão.

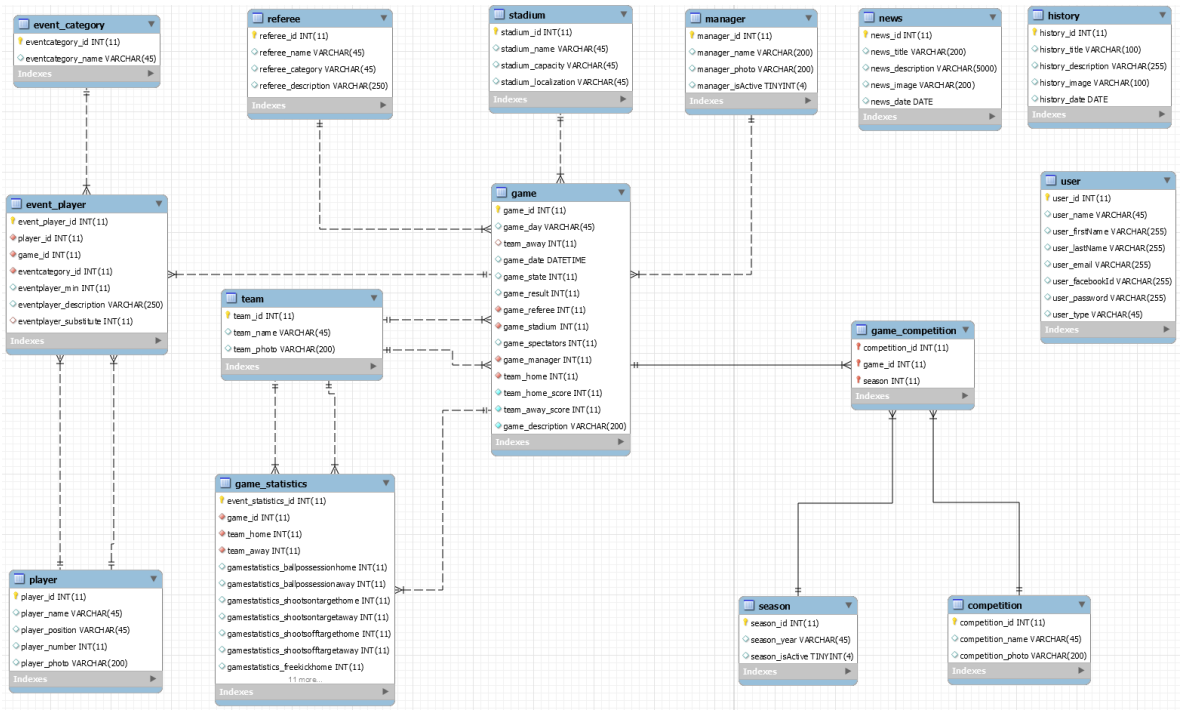


Figura 17.: Modelo ER

Seguidamente vai ser realizada uma breve descrição de cada tabela presente na figura 17, bem como, o objectivo que cada uma vai ter, de modo a garantir as funcionalidades da aplicação *web* para os seus utilizadores.

- ***Competition***: A tabela ***Competition*** vai guardar toda a informação relacionada com as competições que sejam introduzidas pelo administrador;

- ***Event_category***: A tabela ***Event_category*** é responsável por armazenar os eventos presentes na aplicação *web*, tais como golos, cartões, substituições entre outros eventos. Esta tabela foi criada, de forma a poder se adaptar a qualquer desporto e assim guarda qualquer evento de determinado desporto;

- ***Event_player***: A tabela ***Event_player*** é uma tabela intermédia criada, face à relação (N - M) entre as tabelas (***Event_player*** e ***Player***), devido ao facto de um jogador poder estar associado a diversos eventos, em vários jogos. A intenção desta tabela é permitir que se registem todos os eventos que estejam relacionados com um jogador, em determinado jogo. A informação presente nesta tabela vai ajudar a calcular as estatísticas/confrontos de determinado jogador;

- ***Game***: A tabela ***Game*** tem como primordial objectivo armazenar informação sobre o jogo, nomeadamente a jornada a que se refere o jogo, resultado, entre outros. É necessário referir que a própria tabela tem várias relações com diversas tabelas como

competição, estádio, clube, árbitro, entre outras. Assim é possível guardar toda a informação necessária para que um utilizador consulte todos os detalhes de determinado jogo;

- ***Game_competition*:** A tabela ***Game_competition*** irá armazenar informação sobre todos os jogos que o clube fez em determinada competição e época. Esta tabela vai ser a chave para conseguir perceber os jogos em que determinado jogador/treinador participou em dada época ou competição;
- ***Game_statistics*:** A tabela ***Game_statistics*:** tem o objectivo de armazenar a informação relativamente às estatísticas de determinado jogo;
- ***History*:** A tabela ***History*** vai guardar todos as histórias referentes ao clube em questão;
- ***Manager*:** A tabela ***Manager*** tem com o intuito armazenar informação referente ao treinador da equipa, sendo que a mesma tabela vai ser importante para que os utilizadores consultem informação sobre os treinadores que passaram pelo clube;
- ***News*:** A tabela ***News*** permite guardar as notícias que são associadas à equipa, com intuito dos utilizadores visualizarem as mesmas;
- ***Player*:** A tabela ***Player*** é responsável por guardar toda a informação sobre um jogador, como posição, dados pessoais entre outras informações;
- ***Referee*:** A tabela ***Referee*** guarda a informação sobre os árbitros que fiscalizaram um ou mais jogos do clube detentor da aplicação;
- ***Season*** A tabela ***Season*** tem o objectivo de armazenar informação sobre determinado época desportiva;
- ***Stadium*:** A tabela ***Stadium*** armazena a informação sobre os estádios em que a equipa joga ou jogou;
- ***Team*:** A tabela ***Team*** permite armazenar toda a informação referente às equipas que disputam um jogo contra o clube, por exemplo permite ao administrador inserir logo as equipas de todas as competições em que o clube participa em determinada época;
- ***User:*** A tabela ***User*** é responsável por registar a informação sobre o utilizador da aplicação. Esta tabela tem um campo que é o tipo de utilizador, o que permite distinguir os utilizadores que têm ou não permissão para aceder ao *Backend*.

3.2.4 *Casos de uso*

"O diagrama de casos de uso serve essencialmente para especificar as funcionalidades que o sistema disponibiliza aos utilizadores e definir a fronteira do sistema com o ambiente" (Tradução nossa)[2], segundo as palavras dos autores João Miguel Fernandes e Ricardo Machado. No cartão de *volere* está presente um campo onde se define o nome do caso de uso para o requisito em questão.

No que respeita aos casos de uso, foi elaborado um caso de uso para cada requisito funcional, de modo a perceber quem são os responsáveis por cada requisito.

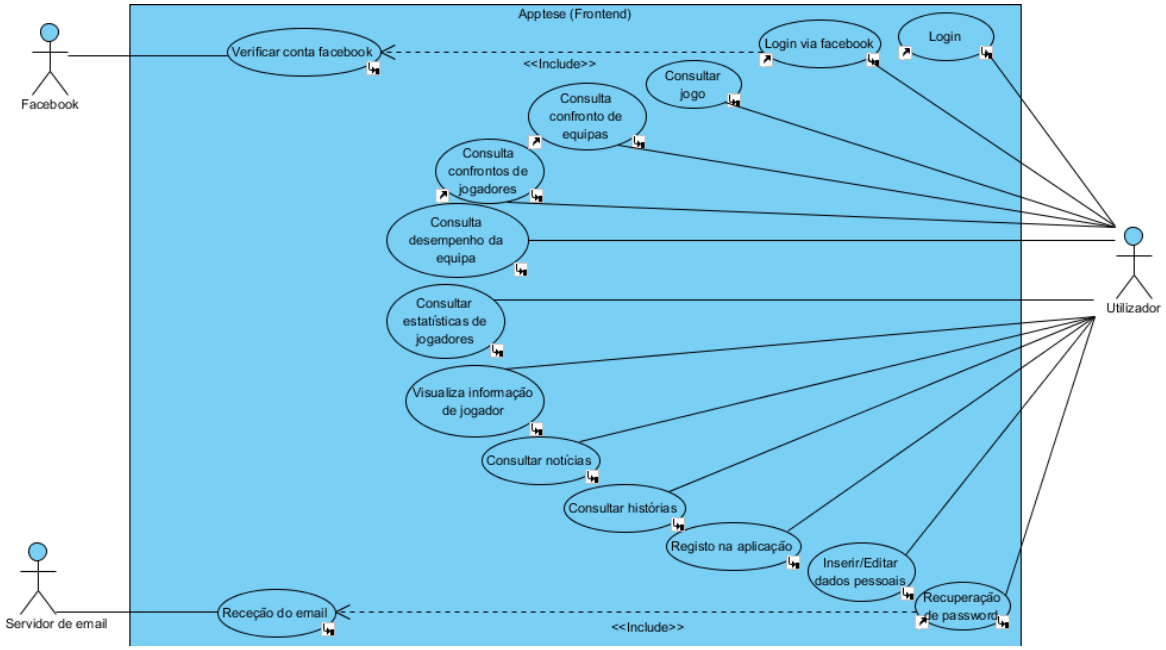


Figura 18.: Diagrama de casos de uso do utilizador (*Frontend*)

Na figura 18 temos presentes todos os casos de uso referentes aos utilizadores da aplicação. Por exemplo quando um utilizador pretende realizar *login* na aplicação, tem a possibilidade de realizar a operação com uma conta criada na aplicação, ou então pela sua conta do *facebook*. Quando o utilizador realiza *login* com a sua conta de *facebook* existe a necessidade de validar as suas credenciais com os servidores da mesma, daí o uso do *include* (Verificar conta *facebook*).

No entanto o utilizador tem ainda a possibilidade de consultar informações referentes ao clube, desde notícias, jogos da sua equipa, bem como, a análise de confrontos e estatísticas da equipa ou dos seus jogadores.

No que respeita ao administrador, a figura 19 apresenta os seus casos de uso em *package*, uma vez que por serem diversos casos de uso existiu a necessidade de serem agrupados.

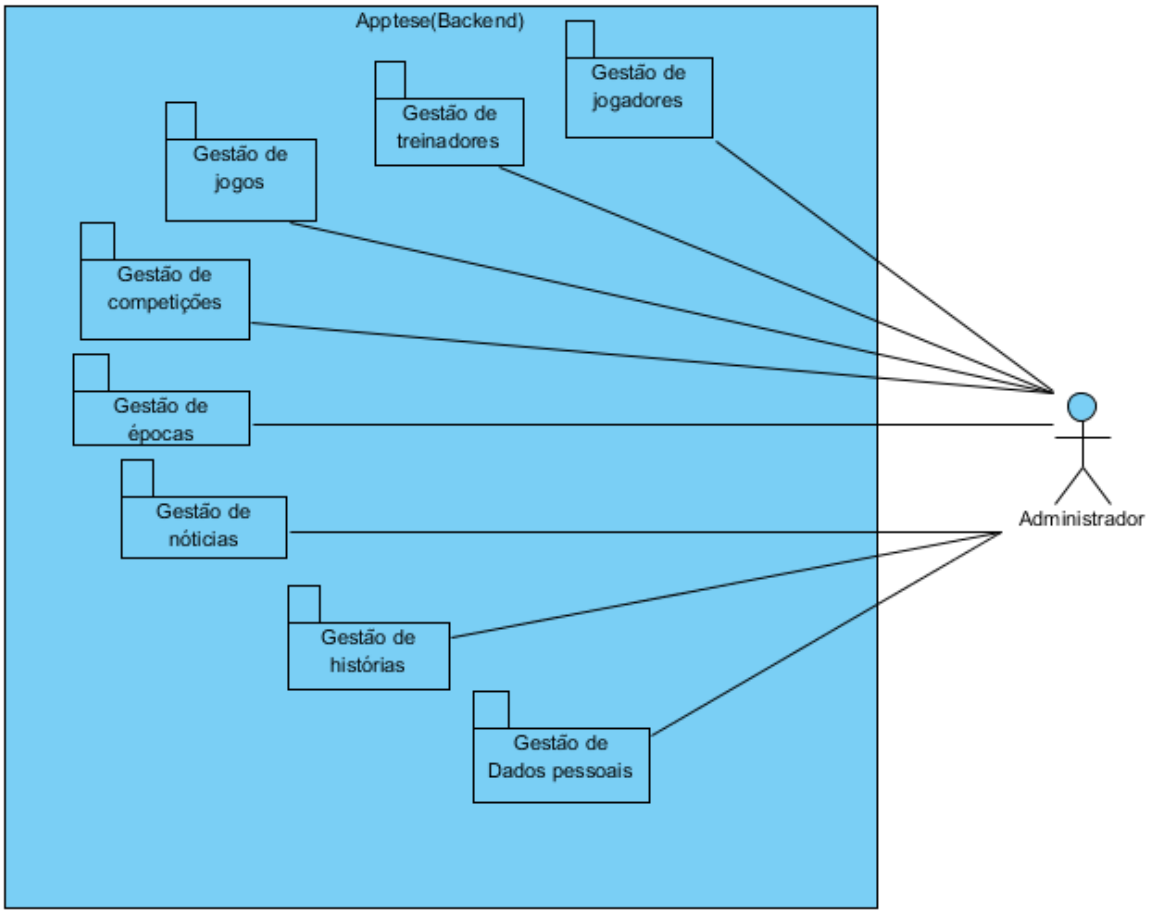


Figura 19.: Diagrama de casos de uso do administrador

É necessário referir que na figura 19, o *package* gestão de jogos, contem também a gestão de equipas adversárias, eventos associados ao jogo, estádios, árbitros e estatísticas.

Com a representação deste diagrama e consequente explicação é possível analisar a sua importância, tal como a dos diagramas seguidamente apresentados. Os diagramas de casos de uso são relevantes para a implementação das funcionalidades.

3.2.5 *Diagramas de actividades*

O diagrama de actividades é responsável por definir o fluxo que a funcionalidade tem que seguir para que o requisito seja concluído como o planeado. Contudo, cada caso de uso vai ter associado um diagrama de actividades.

Na figura 20 será apresentado o diagrama de actividades para a opção de consultar confrontos de um determinado jogador.

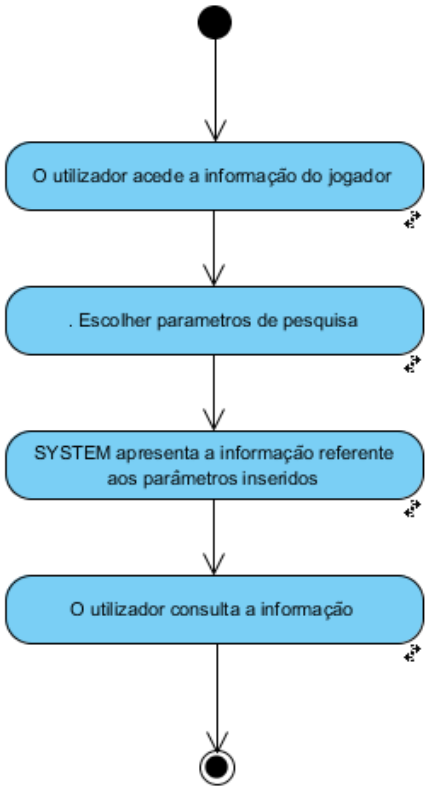


Figura 20.: Diagrama de actividade - Caso de uso (Consulta confrontos de jogadores)

Na figura 20 é possível analisar o diagrama de actividades relacionado com a análise de confrontos de um jogador, realizada pelo utilizador. No primeiro passo, o utilizador terá que aceder à informação do jogador, para que posteriormente defina os parâmetros para que seja feita a sua consulta. Ao fim de definir os parâmetros de pesquisa, o sistema é responsável por obter a informação sobre o que o utilizador pretende, para que depois lhe seja apresentada.

No que concerne a um diagrama de actividades do administrador, foi escolhido o caso de uso inserção de um jogo.

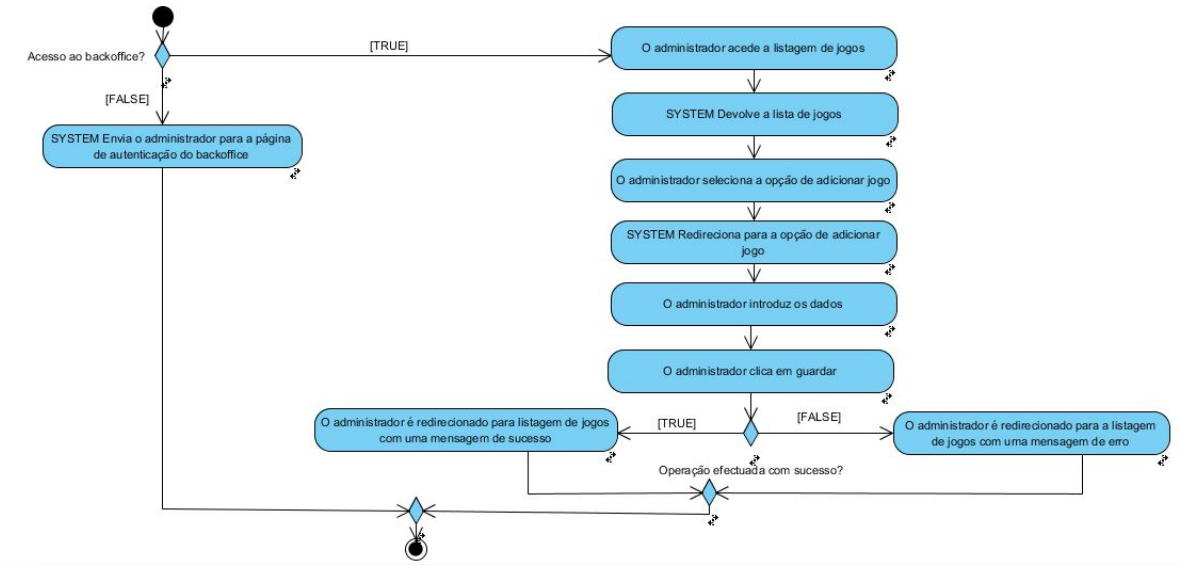


Figura 21.: Diagrama de actividades - Caso de uso (Inserir jogo)

Na figura 21 é possível constatar que no início da operação é validado se o utilizador em questão tem acesso ao *backend*. É uma medida de segurança implementada em todas as funcionalidades dos administradores. Seguidamente à validação da conta, o administrador tem que aceder à listagem de jogos, para que depois tenha a possibilidade de criar o jogo. Por fim, é validada toda a informação introduzida pelo mesmo e depois recebida uma mensagem com a informação associada à operação realizada.

3.2.6 *Diagramas de sequência*

O diagrama de sequência é mais orientado à parte de código, sendo que é um complemento do diagrama de actividades, para perceber quem é que realiza certo pedido. Mais uma vez cada requisito terá o seu diagrama associado, o que permite uma implementação mais simples para o programador, pois só tem que se preocupar em seguir o fluxo pedido pelo o utilizador.

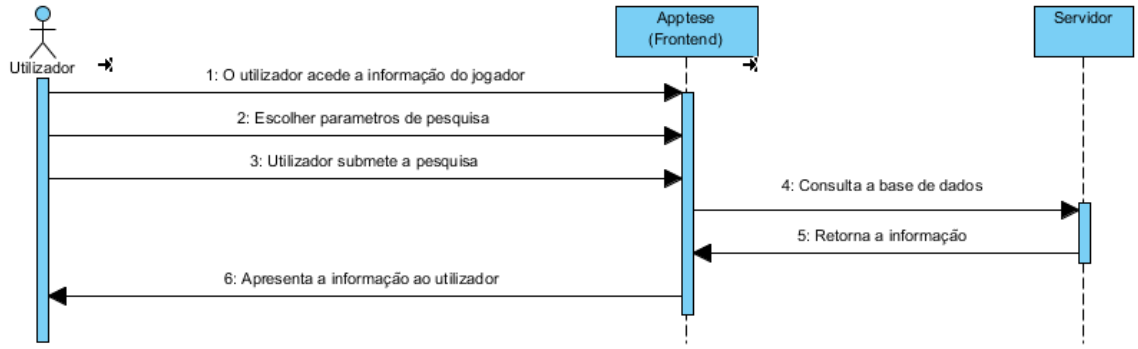


Figura 22.: Diagrama de sequência - Caso de uso (Consulta confrontos de jogadores)

No que respeita a figura 22 é possível verificar o fluxo de pedidos que vão ser desencadeados pelo actor da funcionalidade, no entanto o sistema, vai necessitar de comunicar com o servidor para obter a informação e para que seja construída a resposta a apresentar ao utilizador.

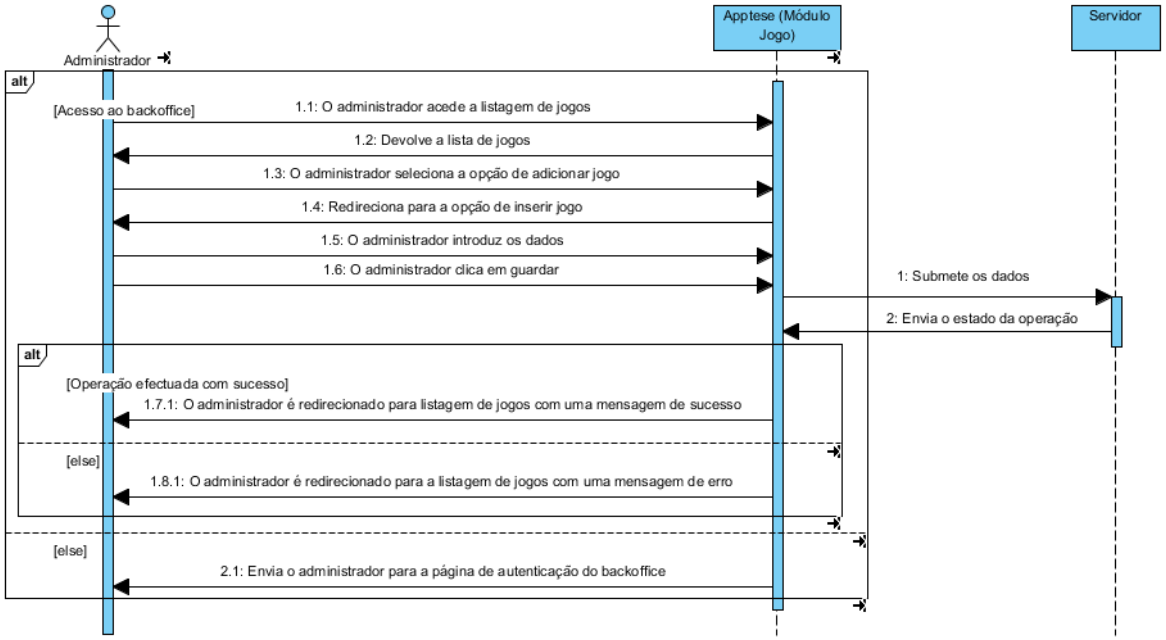


Figura 23.: Diagrama de sequência - Caso de uso (Inserir jogo)

Na figura 23 é apresentado o fluxo que deve ser implementado para que o utilizador consiga introduzir a informação sobre determinado jogo.

O analista tem o critério de decidir o que pretende colocar neste diagrama, ou seja, pode colocar logo o nome dos métodos ou apenas definir os fluxos, deixando assim ao critério do programador a escolha dos métodos a utilizar para que tal funcionalidade seja implementada com sucesso.

3.2.7 *Escolha da arquitectura de software*

No que respeita a arquitectura de *software* a escolha recaiu no cliente-servidor, pois é arquitectura de *software* mais indiciada para projectos *web*. Como foi mencionado no estado de arte (secção 2.3.7.1), esta tecnologia tem enormes potencialidades que têm de ser bem aproveitadas, com o intuito de tirar o maior partido da aplicação. A escolha desta arquitectura, incidiu também sobre o facto de maior parte das aplicações *web* e *websites* utilizarem esta arquitectura, sendo no entanto crucial estudar estratégias para combater os problemas que lhe estão associados. Na secção sobre *performance* 4.5.1 será abordada uma possível solução para combater as desvantagens desta arquitectura.

É necessário salientar que ao operar nesta arquitectura o servidor conterá a aplicação, utilizando um servidor HTTP, de modo a responder aos pedidos feitos pelo cliente via *Browser*.

4

IMPLEMENTAÇÃO, TESTES E DEPLOYMENT

Tal como foi referido no capitulo 3 o processo de desenvolvimento de um sistema de *software* está faseado em várias etapas. Para além da análise e da concepção já anteriormente explanadas, existem ainda as seguintes fases:

- **Implementação**: Este passo é onde se transforma todos os diagramas em código, fazendo com que o produto obedeça às funcionalidades pedidas pelo cliente. Este passo é fundamental para definir quais as tecnologias, *design patterns*, segurança, *performance* entre outras características;
- **Testes**: Depois da implementação é necessário testar toda a implementação, de modo a garantir as funcionalidades.
- ***Deployment***: Por fim, é necessário disponibilizar a aplicação aos utilizadores colocando a aplicação no servidor, com todas as dependências e ferramentas para que o *software* se comporte como no ambiente de desenvolvimento.

4.1 DECISÕES

Antes de iniciar qualquer implementação de um produto de *software*, é necessário efectuar várias decisões, de modo a verificar qual será a tecnologia, *design pattern* entre outras, que melhor se enquadra com o projecto, com objectivo de atingir o melhor desempenho e satisfação por parte do cliente. Estes são dois atributos de qualidade que podem ser associados a um dado projecto, mas nem sempre são estes ou só estes.

Neste tópico será importante analisar os pontos fortes e fracos, de forma a conseguir obter as escolhas mais consensuais e adaptáveis ao produto.

4.1.1 *Tecnologias - Server side*

As tecnologias do lado do servidor são responsáveis por lidar com os pedidos por parte do cliente e são elas que vão interagir com o servidor HTTP, de modo a fornecer os recursos

que o mesmo precisa. A linguagem utilizada também é responsável por interagir com a base de dados, para conseguir consultar a informação e para a gestão da mesma. Para esta escolha foram analisadas duas linguagens: *ASP.NET* e o PHP.

A escolha recaiu sobre o PHP, devido a diversos factores tais como: a sua facilidade de aprendizagem, agilidade, a sua fácil implementação que diminui o tempo de desenvolvimento, a existência de uma grande comunidade e suporte para auxiliar os programadores, facilidade em conseguir escalar uma aplicação, compatibilidade com vários tipos de base de dados, facilidade em colocar num servidor(*deploy*) e ser *open source*. A escolha poderia ter recaído sobre o *ASP.NET*, contudo é uma tecnologia que proporciona um desenvolvimento mais lento que o PHP, não só pelo facto de ser mais difícil de aprender, mas também pelo facto de necessitar de mais código para implementar determinadas soluções. Outras desvantagens do *ASP.NET* são a dificuldade que apresenta para ser colocado num servidor(*deploy*), bem como, os custos que são necessários para conseguir implementar um produto, visto não ser *open source*. Apesar de uma das vantagens do *ASP.NET* ser a sua robustez face ao PHP, foram exploradas técnicas para aumentar a segurança e robustez no PHP.

Em suma, o PHP foi sempre uma solução utilizada para resolver os problemas associados aos produtos *WEB*[16], sendo por isso a linguagem mais utilizada no desenvolvimento daqueles produtos[9].

Neste produto foi utilizado o PHP 7, que é a última versão da linguagem. Esta versão teve melhorias significativas, nomeadamente a nível de desempenho, sendo pelo menos duas vezes mais rápido que a versão anterior e consumindo menos memória. As funções de *MySQL* para comunicar com a base de dados foram removidas[17], obrigando assim os programadores a utilizar os *PHP Data Object (PDO)*. Os PDO's são instâncias responsáveis por fazer as comunicações com a base de dados, o que permite reduzir o SQL *Injection*. No nosso caso vamos utilizar o *Object Relational Mapping (ORM)* da *framework*[18], que é escrito em *Zephir/C*, o que proporciona uma enorme *performance* na comunicação com a base de dados.

Outros dos pontos importantes desta nova versão do PHP é a implementação tipificada das variáveis, algo que nunca foi utilizado em PHP.

4.1.2 *Tecnologias - Client side*

No que respeita às tecnologias que vão operar no lado do cliente, as escolhidas foram o HTML, CSS, *JavaScript* e AJAX, nas suas últimas versões, mais concretamente HTML5 e CSS3.

Muitos projectos utilizam diversas *frameworks* ou *plugins* associados ao lado do cliente.

Neste projecto foram utilizadas as seguintes:

- ***jQuery***: A escolha do *jQuery* foi essencial para tirar partido do *JavaScript* de uma forma mais limpa e eficaz. Esta *framework* oferece uma tremenda facilidade na utilização do *JavaScript*. O *jQuery* foi essencial na utilização dos pedidos AJAX entre a manipulação de elementos HTML.

- ***Bootstrap***: O *bootstrap* é uma *framework* com base em CSS, *JavaScript* e outras tecnologias, que tem como objectivo adaptar uma solução *web* a qualquer dispositivo. Como todos sabemos um portátil não tem as mesmas resoluções que um *smartphone* e é aqui que o *bootstrap* demonstra as suas capacidades, conseguindo adaptar o nosso produtos a várias resoluções.

- **DataTables**: *DataTables* é um *plugin* que é muito utilizado pela comunidade *web* para apresentar a informação em tabelas, com diversos elementos, como paginação, ordenação ou até mesmo pesquisa. Este *plugin* foi muito utilizado na parte das listagens no *backend*, o que ajudou na sua implementação.

- ***jQuery Validation***: O *jQuery Validation* foi crucial para tratar os dados inseridos pelo utilizador, ou seja, alertá-lo para campos obrigatórios, ou mesmo para campos que só possam ter um determinado formato (números, datas entre outros). Este *plugin* permite filtrar o que o utilizador insere e tem mais vantagens que as validações oferecidas pelo HTML5, daí o motivo da sua utilização. É necessário salientar que no caso desta aplicação os dados são validados quer no lado do servidor, quer no lado do cliente, tornando-se mais dinâmicos no lado do cliente e mais prático para o utilizador da aplicação.

- **Charts**: Este *plugin* foi utilizado para conceber os gráficos na aplicação, ou seja, só é necessário aplicar o tipo de gráfico que pretendemos e enviar os dados para popular o gráfico, cabendo ao *plugin* a responsabilidade de apresentar os mesmos na aplicação.

- ***Facebook***: O *facebook* foi utilizado para possibilitar que os utilizadores consigam autenticar-se com a sua conta daquele rede social. Neste caso é utilizado o protocolo de *OAuth2*[1], ou seja, as duas aplicações comunicam entre si e o *facebook* é responsável por enviar a informação do utilizador em questão. Com esta abordagem é possível fazer autenticações em diversas aplicações sem ter os dados do cliente.

4.1.3 *Base de dados*

O SGBD é um dos elemento fulcrais para qualquer aplicação que guarde informação, com o objectivo de sustentar os conteúdos que a aplicação utiliza.

1 Página web: `https://oauth.net/`

No caso da aplicação a desenvolver vai ser necessário utilizar dados relacionais, face à informação que é necessário relacionar, este foi o aspecto que levou a utilizar o *MySQL* em vez do *MongoDB*, e também a facilidade de utilizar o *MySQL* nos servidores.

Nesta aplicação irão existir diversas *querys* que vão utilizar o *join*[2], algo que o *MongoDB* não suporta visto não ser relacional. O *MySQL* tem um excelente comportamento com *querys*/transações de grande complexidade, tendo uma enorme facilidade em tratar dados em grande dimensão, devido a forma como os armazena.

Em relação ao *store engine* foram analisadas duas *store engines: InnoDB* e *MyISAM*. A escolha recaiu no *InnoDB* devido ao facto de suportar chaves estrangeiras, funcionalidade essa que vai ser necessária na aplicação. Este *store engine* permite ainda transações, ou seja, permite tirar partido do *commit* quando se pretende guardar a transação ou fazer *rollback* quando o produto detectar que algo correu mal.

No entanto, esta escolha vai depender sempre do que o produto necessitar, em termos de robustez e integridade dos dados e desempenho e tipos de operação que se vão realizar na aplicação.

Em suma, esta escolha deve sempre ter em base as necessidades e requisitos do projecto. A escolha recaiu no *MySQL* face ao seu desempenho, segurança e rapidez de integração com o PHP. Existem diversas formas de optimizar o *MySQL*, aspectos que serão referenciados no tópico da *performance* (4.5.2).

4.1.4 *Framework PHP*

No que concerne à escolha da *framework*, optou-se pela escolha da *framework PhalconPHP*. Esta escolha baseou-se muito nas características que a mesma oferece, bem como, na sua excelente *performance* face às outras *frameworks* PHP. Esta é escrita na linguagem C, o que permite uma melhor optimização, tal como se retira das figuras abaixo (Fig.24 e Fig.25).

2 O *join* permite que duas tabelas sejam relacionadas

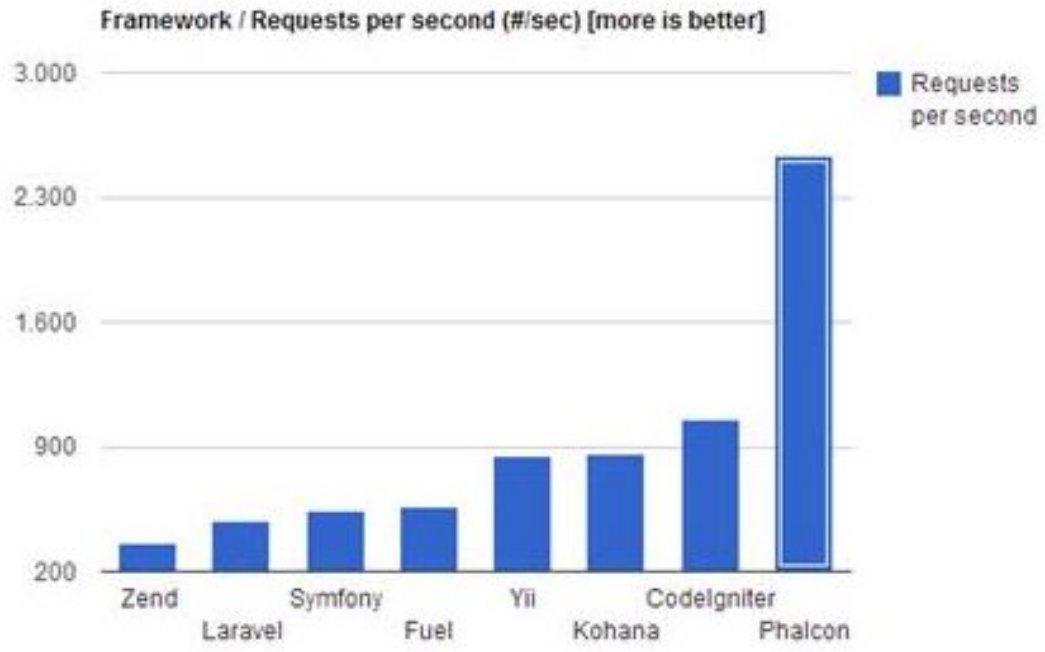


Figura 24.: *Framework PhalconPHP* pedidos por segundo [19]

Na figura 24 é possível verificar que o *Phalcon* destaca-se das outras *frameworks*, atingindo um grande número de pedidos por segundo. Estes dados são interessantes para garantir a melhor escalabilidade [3] possível na aplicação a desenvolver. A figura 25 ilustra o tempo de resposta aos seus pedidos, fazendo jus à sua rapidez face as outras *frameworks* [19].

3 Um dos requisitos para aplicação é garantir a escalabilidade, referente ao requisito 43 constante dos anexos (A)

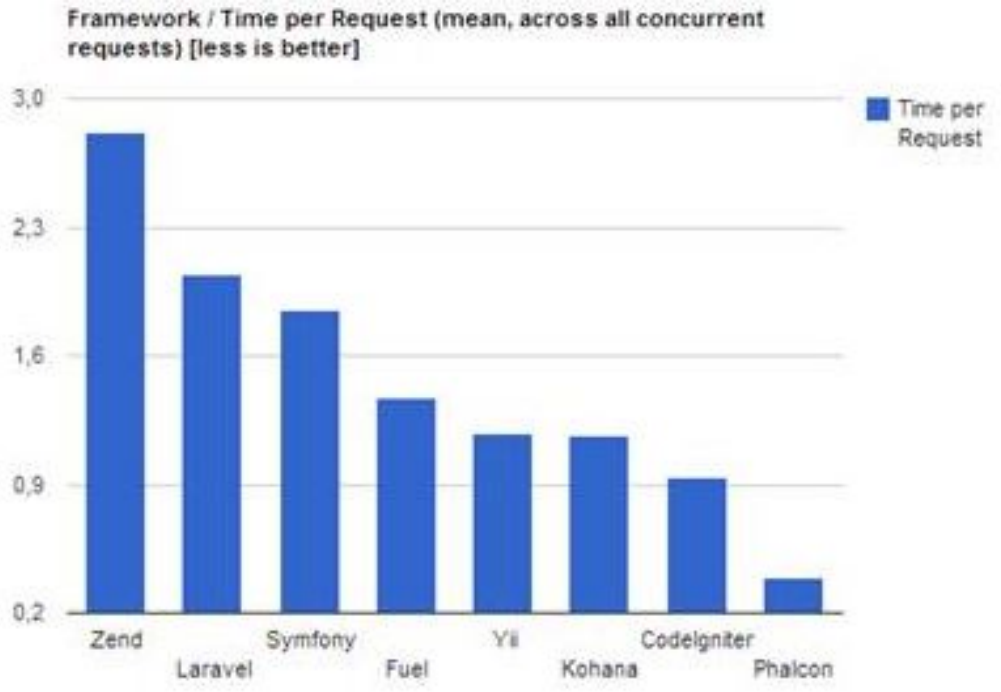


Figura 25.: *Framework PhalconPHP* tempo de resposta

É de referir que todas as *frameworks* usam a mesma arquitectura de *software*, *design pattern* MVC, o que permite que os programadores PHP se adaptem facilmente a qualquer uma. No que respeita às consultas à base de dados, todas as *frameworks* dispõem de um ORM de forma a auxiliar o uso do paradigma OOP, destacando-se a *framework* escolhida em certos aspectos:

- **HMVC**: Suporte do HMVC permite criar vários módulos MVC, e assim divide as funcionalidades por módulos. No caso da aplicação a desenvolver, existirá o módulo *frontend* e *backend*[4].
- ***Cache***: O *phalcon* disponibiliza um sistema de *Cache*[5] para as *views* que são apresentadas aos utilizadores, bem como, para as consultas que são feitas à base de dados. Esta técnica permite reduzir *overhead* e pedidos à base de dados.
- ***Access Control List (ACL)***: Esta técnica possibilita limitar as funcionalidades da aplicação, consoante o perfil/permissões do utilizador;
- ***Micro Application***: É utilizado para criar *web services*, o que facilita a criação dos mesmos;

4 Estes módulos vão ser descritos com mais detalhe na secção 5.1
5 Permite armazenar informação de forma a aumentar a *performance*

- ***Template Engine***: Esta *framework* tira partido do *volt*[6], um *template engine* muito eficaz que permite manusear e integrar informação nas *views* que são enviadas pelo *controller*;

- **Segurança**: Disponibiliza diversos métodos que permitem aumentar a segurança da aplicação a desenvolver.

Estes pontos foram determinantes para a escolha da *framework*, sendo ainda fundamental a leitura da documentação[20] da mesma para uma aprendizagem mais aprofundada, nomeadamente o livro[21] que permitiu optimizar aplicação e tirar partido das vantagens que a *framework* oferece.

Em suma, poderia ser escolhida outra *framework*, mas optou-se pela escolha desta com o intuito de explorar e aprender uma nova *framework*. A escolha não foi apenas com base no seu desempenho mas também para explorar uma *framework* que tem por base a linguagem C que é conhecida pelos seus grandes níveis de optimização.

4.1.5 *Escolha de servidor HTTP*

O servidor HTTP é o elemento chave para qualquer aplicação que opere sobre o protocolo HTTP. Como referenciado na secção 2.4.5, este elemento é responsável por tratar os pedidos dos utilizadores, associando o *url* a determinada aplicação. Com a interpretação do *url*, o servidor é responsável por solicitar os requisitos à aplicação, de modo a obter os recursos para apresentar o resultado esperado pelo cliente.

No que respeita a soluções *web* temos dois servidores HTTP, sendo eles o *Apache* e o *Nginx*. O *Apache* é o servidor HTTP mais utilizado a nível de soluções *web*[6], sendo que o *Nginx* tem vindo a ganhar terreno, face aos poucos recursos que consome. Em termos de segurança, o *Apache* garante mais segurança e eficácia. Em termos de *performance*, o *Nginx* obtém melhores resultados quando se trata de uma solução com conteúdo estático, ou seja, não varia muito, existindo poucas mudanças face aos pedidos do cliente. No entanto, quando se fala de conteúdo dinâmico eles obtêm o mesmo nível de *performance*.[22] (Capítulo 3, p.41-51)

O *Apache* também tira partido da sua simplicidade de configuração, sendo até o servidor HTTP mais utilizado nas plataformas *web* existentes. Resumindo, a escolha foi para o *Apache* visto ser um servidor HTTP robusto e ser maioritariamente utilizado pela comunidade *web*, no entanto serão abordadas algumas técnicas que permitem aumentar a *performance* da aplicação, alterando certos parâmetros na configuração do mesmo.

6 Página web: `https://docs.phalconphp.com/ar/3.2/volt`

4.1.6 *Design patterns*

O *design pattern* escolhido foi o MVC, pois permite organizar a aplicação de uma forma *standard*, com o intuito de se adaptar a novos elementos que possam participar no projecto. A *framework* escolhida também tira partido deste padrão de desenho, o que facilitou a integração do mesmo.

Para a utilização deste *design pattern* foi necessário configurar as rotas para que aplicação saiba quem é o *controller* responsável por responder ao pedido do utilizador. A utilização deste padrão permite separar as funcionalidades, ou seja, o *model* é responsável por ter a lógica do negócio e fazer os pedidos à base de dados. O controlador apenas irá fazer de intermediário entre o *model* e *view*, sendo o responsável por pedir a informação ao *model* e enviar para a *view*. A *view* é responsável por mostrar a resposta ao utilizador.

4.1.7 *API*

No que respeita à escolha da API, a opção recaiu sobre o REST, visto tirar partido do JSON o que torna mais fácil a comunicação com outros serviços, e visto a *framewrok* escolhida ter módulos que facilitam a implementação da mesma.

Neste produto vai ser utilizada a API do *facebook* para permitir o *login* com conta de *facebook*, ou seja, é possível validar as credenciais de determinada conta sem ter os dados do cliente, sendo que neste caso é utilizado o protocolo *OAuth2*.

Foi também desenvolvida uma API em REST, para que mais tarde a aplicação consiga comunicar com outras, podendo até vender informação a outras aplicações. Solicitou-se uma colaboração com o zerozero para poder, por exemplo, importar informação de jogos e jogadores, facilitando assim a introdução de informação por parte do administrador, mas nunca se obteve uma resposta.

Em relação à escolha do tipo de API, a decisão recaiu sobre o REST, visto ser uma API mais fácil de implementar e que tem a vantagem de tirar partido do protocolo HTTP, conseguindo assim evitar o uso de protocolos de transporte. O SOAP é utilizado em situações mais complexas, no caso de transações devido a sua robustez na construção da sua mensagem. Outra vantagem do REST é poder enviar respostas em JSON ou XML, no caso do SOAP só utiliza XML.

O REST tem um óptimo desempenho a lidar com diversos *request* e tratamento dos pedidos desencadeados pelos clientes.

4.2 SEGURANÇA

Na implementação deste produto foi crucial analisar e estudar medidas de tornar aplicação mais segura e robusta. Foram analisados alguns problemas que estão associados às vulnerabilidades das aplicações *web*, para isso foram consultados alguns livros[23][24][25], bem como, o OWASP, com a finalidade de implementar medidas preventivas.

Nos tópicos serão abordadas soluções introduzidas na aplicação, para proteger a mesma destas falhas, e com isso demonstrar que o PHP afinal também pode ser parte integrante duma solução tecnológica segura.

4.2.1 *SQL injection*

No que respeita ao SQL *injection*, foram utilizadas medidas definidas pela OWASP, de modo a prevenir este ataque. A primeira abordagem a este ataque foi aplicar o ORM que a *framework PhalconPHP* oferece. Nos casos em que a aplicação não usufrui do ORM(7) será utilizada uma classe para realizar as *querys* à base de dados que também previne os ataques.

Posteriormente foi necessário criar um mecanismo para todos os parâmetros utilizados, para realizar uma filtragem de todos os dados inseridos na base de dados, bem como, os dados utilizados para qualquer consulta à base de dados.

Com estes passos foi possível evitar o SQL *injection*, sendo de referir que o PHP 7 já deixou de utilizar as funções *MySQL*[17] do PHP que permitiam SQL *injection*. Este ataque ocorria mais nas primeiras versões das funções *MySQL* disponibilizadas pelo PHP.

4.2.2 *CSRF*

Na secção 2.4.12.2 foram mencionados os detalhes deste ataque e como o evitar. No caso da nossa implementação adoptamos a medida que foi apresentada anteriormente, ou seja, gera-se um *token* aleatório em todos os formulários. Com esta solução não é possível replicar este ataque, pois foi implementado um método na parte do servidor, que vai sempre validar o *token*.

Assim sendo, é garantido que mesmo que um intruso pretenda submeter ou realizar um pedido por parte de um utilizador, ele nunca vai conseguir, pois o *token* é aleatório e validado antes de fazer qualquer acção.

No entanto é preciso realçar que a *framework* utilizada tem já métodos que evitam este ataque, e por isso foi utilizado este modelo, uma vez que a *framework PhalconPHP* tem uma componente de segurança muito robusta e eficaz.

7 Este motivo será abordado na secção 4.5.2

4.2.3 *XSS*

Este ataque tem como principal objectivo introduzir *JavaScript* no *website* que pode ter como finalidade obter dados do utilizador, ou fazer com que por exemplo corra determinado código para alterar o comportamento do *website* e posteriormente o do servidor.

Para resolver este possível ataque foram utilizados métodos que permitem fazer uma limpeza a todo o código que possa ser malicioso, de modo a evitar tal problema. Na parte da manipulação de enviar dados para a *view* ou para a base de dados é utilizado um filtro que é disposto pela *framework PhalconPHP*, para evitar que determinados caracteres sejam introduzidos. No caso da *framework PhalconPHP*, quando é realizado um *request*, ao obter as variáveis, é possível definir o tipo da variável que estamos a obter. Esta foi a técnica utilizada para garantir que a aplicação não fica exposta a esta vulnerabilidade.

4.2.4 *SSL*

Na implementação da aplicação foi utilizado um certificado SSL, de forma a encriptar a informação que é enviada entre o cliente e servidor e vice-versa. Para sabermos se determinado site utiliza SSL, o endereço tem que começar por *Hypertext Transfer Protocol Secure (HTTPS)*, sendo que precisa de ter um cadeado e a palavra "seguro", pois é necessário verificar se a entidade certificadora é fidedigna.

O funcionamento do SSL é muito simples, ou seja, quando o cliente acede a um *website* que utilize SSL, vai pedir o certificado, depois de obter o certificado e verificar que a entidade certificadora é fidedigna. Posteriormente o cliente cria uma chave, denominada de chave pública e envia ao servidor, chave essa que vai ser essencial para que seja desencriptada toda a informação.

Esta segurança é mais utilizada em negócios com mais risco, por exemplo bancos ou *websites* de compras e pagamentos *online*, sendo que nunca é demais utilizar este mecanismo porque transmite mais segurança aos utilizadores de qualquer aplicação.

4.3 Regras de programação

Em geral, as regras foram criadas para permitir que existisse um padrão referente a qualquer tarefa, tornando o código mais genérico e fácil de entender entre uma equipa de desenvolvimento ou mesmo para um novo elemento que tenha entrado recentemente numa equipa de desenvolvimento.

Nesta fase vão ser abordadas algumas das regras que estão inerentes ao desenvolvimento *web*, pois existem variadíssimas normas de programação referentes a todas as linguagens de programação. No desenvolvimento *web* quando existem páginas que se repetem, ou parte

delas, convém incluir essas mesmas partes num ficheiro, por exemplo o menu, *header*, *footer* e o *sidebar* da aplicação/*webiste* porque normalmente são constantes em todas as páginas *web* e isso aumenta a modularidade, e facilita a manutenção do código, pois se existir um erro apenas se corrige num ficheiro.

É fundamental incluir todos os ficheiros CSS referentes aos estilos da aplicação *web*, no *header* da aplicação, sendo carregados primeiro por estarem relacionados com o *design* do *Website*. É necessário salientar que os ficheiros *JavaScript* que estão relacionados com determinados comportamentos de elementos *web* devem ser incluídos no *footer*. Os ficheiros responsáveis pelo CSS não causa tanto *overhead* como os de *JavaScript*, havendo um carregamento mais rápido da aplicação *web*. O carregamento dos estilos referentes a determinadas classes ou *id's* de elementos HTML, é fulcral para que depois o *JavaScript* os consiga interpretar sem qualquer erro.

Estas são algumas das normas importantes que diversos programadores utilizam para permitir o aumento da *performance*, bem como, diminuir o *load* da aplicação *web*.

4.4 CONVENCÇÕES DE PROGRAMAÇÃO

As convencções de programação, são normas que os programadores devem seguir, de modo a seguirem um padrão no seu desenvolvimento. No caso do PHP a mais conhecida é o *PHP Standard Recommendation (PSR)*, existindo diversas versões, e sendo cada uma apropriada para determinadas partes do código. O mais utilizado é o PSR-2 na parte da concepção do código da aplicação. No que respeita ao PSR-2 podemos verificar a diferença no seguinte código.

```
//Exemplo com PSR-2
  $nome   = "Henrique Sobral";
  $cidade = "Lisboa";

if ($cidade == "Lisboa") {
    echo "Vive em Lisboa";
  } else {
    echo "Nao vive em Lisboa";
  }

// Exemplo sem convenccao
$nome = "Henrique Sobral";
$cidade ="Lisboa";

  if($cidade=="Lisboa")
      echo "Vive em Lisboa";
  else
      echo "Nao vive em Lisboa";
```

Listing 4.1: Exemplo do PSR-2 PHP

É possível verificar que o PSR-2 permite ter um código mais bem estruturado, e uma melhor leitura do mesmo. No que respeita à atribuição de variáveis, o = deve ficar alinhado. Em relação aos ciclos *if*, *for*, *while* deve-se formatar como o *if* presente no código, ou seja, deixar os espaços de forma a proporcionar uma leitura mais facilitada. Atendendo a esta convencção existe uma limitação em relação aos caracteres de cada linha de código, estando aquelas limitadas a 120 caracteres.

O *camel case* permite escrever um nome de uma variável ou função, evitando os espaços, que consequentemente originariam um erro, então o *camel case* o que faz é converter o espaço numa letra maiúscula. Um exemplo disso é como iríamos atribuir o nome da variável nome completo, como se pode constatar no seguinte excerto de código.

```
$nomecompleto       = "Henrique Sobral"; // sem Camel case
$datadenascimento   = 25/02/1992;        // sem Camel case
$nomeCompleto       = "Henrique Sobral"; // com Camel case
$dataDeNascimento   = 25/02/1992;        // com camel case
```

Listing 4.2: *CamelCase*

Como é possível verificar, esta convencção tem muita utilização e ajuda a associação de métodos e atributos das classes. Esta convecção pode ser utilizada em qualquer linguagem de programação.

Em suma, a utilização de convencções ajuda na codificação, ou seja, é seguida uma norma na escrita do código, bem como, a limpeza apresentada no código desenvolvido. No entanto é importante seguir uma convencção para que o código esteja normalizada e com isso evita-se problemas na integração de novo código.

4.5 DESEMPENHO

Quando se desenvolve um produto, é necessário aplicar optimizações no mesmo, possibilitando assim que o produto dê respostas da melhor forma. Nos seguintes tópicos vão ser abordadas algumas estratégias que foram implementadas. Para esta análise foi utilizada uma ferramenta de *profiler*, de modo a perceber onde é que aplicação consumia mais recursos, para ir melhorando o tempo de resposta.

Contudo foi ainda usado um utilitário *ApacheJMeter* que possibilita analisar o tempo de respostas da aplicação, bem como, a escalabilidade associada à aplicação. Seguidamente vão ser abordadas algumas técnicas que possibilitam, obter melhorias de *performance*.

4.5.1 *Load Balancer*

Esta técnica é fundamental para não sobrecarregar um servidor, permitindo ter um *load balancer* que funcionará como uma *proxy*, ou seja, uma barreira que vai decidir para onde encaminhar os clientes.

Esta técnica possibilita que a *performance* e a escalabilidade do *website* sejam incrementadas, conseguindo assim aumentar os recursos aos utilizadores. Na figura 26 é possível verificar uma abordagem para combater a sobrecarga do servidor.

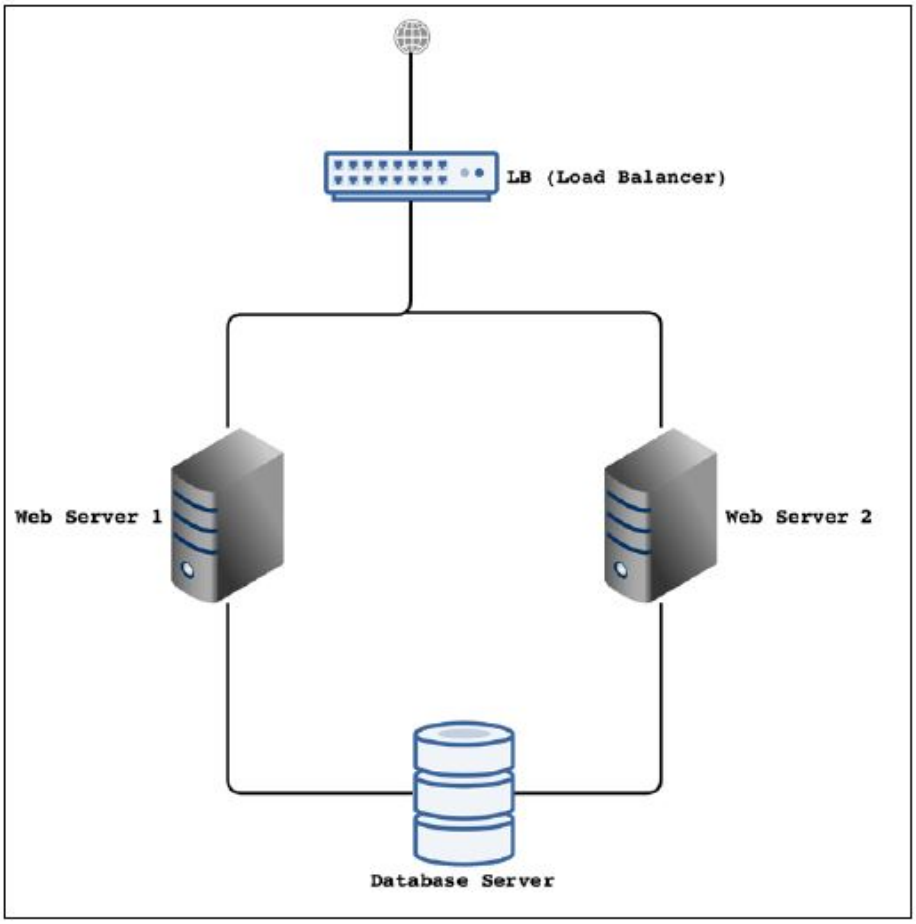


Figura 26.: *Web servers load balancer*
[22](Capítulo 3, p.66)

Na opinião do autor *Altaf Hussain* "Se a nossa aplicação correr apenas num servidor, a *performance* será tremendamente afectada. Também não é boa ideia aplicação correr apenas num único servidor, caso fique em baixo a nossa aplicação ficará inacessível"(Tradução nossa)[22](Capítulo 3, p.66). Contudo é necessário referir que a peça chave desta solução

é o *load balancer*, que vai ser responsável por verificar a carga de cada servidor, de modo a reencaminhar os pedidos para o servidor com mais recursos livres.

No entanto é necessário referir que o servidor onde temos a aplicação alojada, não nos disponibiliza ferramentas para tal solução, devido ao facto de ser gratuito. Na secção *Deployment* 4.8 serão descritos os motivos pela escolha do servidor.

4.5.2 *SQL*

No que respeita ao SQL, existem algumas medidas que possibilitam logo tirar partido de um melhor desempenho da aplicação. Um grande problema é quando se fazem consultas com várias junções de tabelas e é utilizado o *, vai ser devolvida informação que não é necessária ou que maior parte das vezes é repetida, provocando um maior *overhead* na aplicação.

Existem mais normas a seguir quando utilizamos o *MySQL*, de modo a garantir uma melhor *performance* da aplicação, por exemplo, quando queremos realizar pesquisas por um nome, ou por uma descrição devemos utilizar *indexes*. O uso desta técnica permite que as consultas sejam mais rápidas, sendo que o aconselhado é fazer pesquisas por *id's*, visto já serem *indexes*, ou *Primary keys* ou *Foreign keys*. O uso de *indexes* permite que o servidor de base de dados, consulte menos registos o que justifica o seu acréscimo no desempenho.

A *framewrok* escolhida oferece um ORM que permite auxiliar os programadores na realização de *querys* à base de dados, uma vez que já tem métodos pré-definidos para determinadas opções. No entanto o ORM contem informações desnecessárias (dependendo dos casos), o que provoca um aumento no tempo de resposta, que poderá ser prejudicial para o desempenho. Assim, implementou-se uma classe para realizar determinadas *querys*, em que a *performance* tinha de ser garantida (Lista de jogos, Análise de confrontos).

4.5.3 *Profiling*

Quando é desenvolvido um produto de *software*, é preciso verificar os recursos gastos pelo sistema, de modo a garantir o melhor desempenho na execução das suas funcionalidades.

Para o desenvolvimento foi utilizado o *Xdebug*, para despistar erros que ocorriam no processo de desenvolvimento, bem como, garantir a melhor *performance* da aplicação. O *Xdebug* é uma extensão que possibilita aos programadores de PHP tirarem partido das suas preciosas funcionalidades[26].

A figura 27 permite verificar um esquema de *profiling* da aplicação, onde é possível observar o tempo gasto em determinadas operações. Para que fosse possível tal facto foi usado o utilitário *kcachegrind*[8].

8 Pagina web: `http://kcachegrind.sourceforge.net/html/Documentation.html`

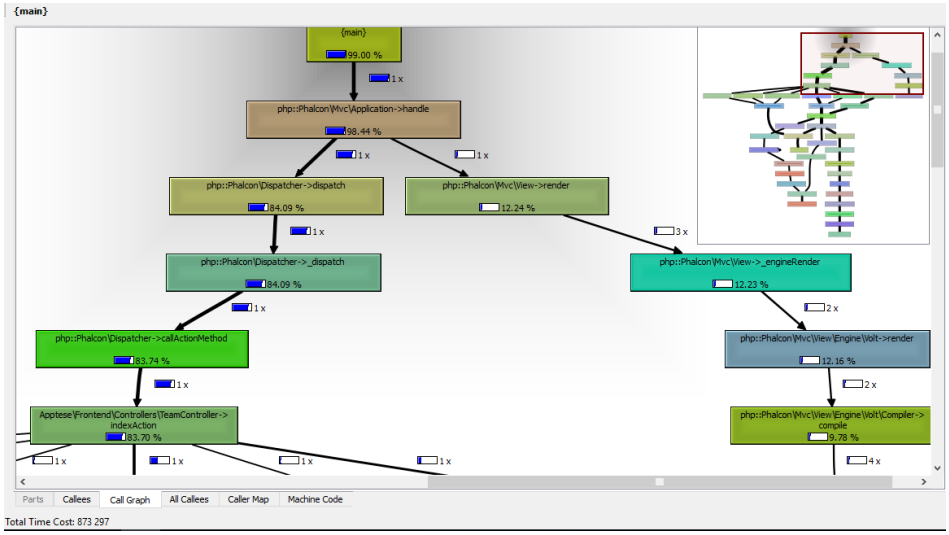


Figura 27.: *Profiling* da aplicação - Funcionalidade listar equipa, jogos e treinadores

A figura 27 representa o *profiler* da listagem de todos os elementos da equipa, bem como, a informação dos jogos referentes a uma época. Este *profiler* permite visualizar quais as operações que aplicação realiza para responder ao pedido do utilizador, permite analisar onde é que a aplicação está a gastar mais recursos. No nosso caso gasta aproximadamente 83% do tempo a realizar pedidos à base de dados, de forma a construir a informação a apresentar ao utilizador. O resto do tempo é gasto na construção da página para o utilizador consultar a informação.

É necessário ainda realçar que aplicação demorou cerca de 0,87 segundos a executar este processo, no entanto este pedido ficou com este tempo porque tem que realizar o *profiler* referente ao pedido, de modo a conseguir perceber se aplicação está a fazer algo de errado, prejudicando a *performance* da mesma.

Em suma, é possível verificar que esta técnica pode ter grande impacto, dependendo também dos requisitos das aplicações, mas é sempre bom analisar este tipo de gráficos, de forma a entender se existe algum problema associado à *performance*.

4.5.4 *ApacheJMeter*

O ApacheJMeter[9] é uma ferramenta que permite analisar diversas questões relacionadas com a aplicação. Na solução desenvolvida foi testada a escalabilidade da mesma, colocando determinados utilizadores (virtuais) a aceder a uma funcionalidade num determinado espaço de tempo.

9 Página web: `http://jmeter.apache.org/`

Esta ferramenta tem diversas funcionalidades, no entanto é necessário referir que foram apenas explorados a escalabilidade e *performance* associadas à aplicação, devido à curva de aprendizagem extensa desta ferramenta.

Em relação ao teste realizado à aplicação desenvolvida, foi realizado um cenário em que 80 utilizadores vão aceder à informação de um jogo num período de 100 segundos, sendo preciso relembrar que no que respeita a informação do jogo vai corresponder diversa informação, desde os jogadores que disputaram a partida, aos eventos e estatísticas referente àquele. Na figura 28 é possível verificar os testes que foram realizados.

Samp...	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	19:36:24.398	Thread Group 1-1	HTTP Request	408		30078	136	357	269
10	19:36:29.798	Thread Group 1-5	HTTP Request	91		30078	136	90	0
100	19:37:26.101	Thread Group 1-50	HTTP Request	100		30078	136	97	0
101	19:37:26.892	Thread Group 1-51	HTTP Request	414		30078	136	361	274
102	19:37:27.306	Thread Group 1-51	HTTP Request	97		30078	136	95	0
103	19:37:28.142	Thread Group 1-52	HTTP Request	390		30078	136	340	245
104	19:37:28.532	Thread Group 1-52	HTTP Request	105		30078	136	103	0
105	19:37:29.404	Thread Group 1-53	HTTP Request	369		30078	136	319	241
106	19:37:29.774	Thread Group 1-53	HTTP Request	108		30078	136	105	0
107	19:37:30.643	Thread Group 1-54	HTTP Request	442		30078	136	389	247
108	19:37:31.085	Thread Group 1-54	HTTP Request	89		30078	136	86	0
109	19:37:31.893	Thread Group 1-55	HTTP Request	385		30078	136	330	238
11	19:36:30.643	Thread Group 1-6	HTTP Request	382		30078	136	329	248
110	19:37:32.278	Thread Group 1-55	HTTP Request	103		30078	136	99	0
111	19:37:33.143	Thread Group 1-56	HTTP Request	469		30078	136	419	250
112	19:37:33.613	Thread Group 1-56	HTTP Request	149		30078	136	148	0
113	19:37:34.393	Thread Group 1-57	HTTP Request	709		30078	136	657	572
114	19:37:35.102	Thread Group 1-57	HTTP Request	163		30078	136	158	0
115	19:37:35.644	Thread Group 1-58	HTTP Request	403		30078	136	350	246
116	19:37:36.047	Thread Group 1-58	HTTP Request	120		30078	136	116	0
117	19:37:36.893	Thread Group 1-59	HTTP Request	482		30078	136	429	245
118	19:37:37.375	Thread Group 1-59	HTTP Request	112		30078	136	107	0
119	19:37:38.143	Thread Group 1-60	HTTP Request	390		30078	136	339	253
12	19:36:31.025	Thread Group 1-6	HTTP Request	95		30078	136	93	0
120	19:37:38.533	Thread Group 1-60	HTTP Request	134		30078	136	129	0
121	19:37:39.393	Thread Group 1-61	HTTP Request	396		30078	136	344	249
122	19:37:39.789	Thread Group 1-61	HTTP Request	83		30078	136	81	0

Scroll automatically? Child samples? No of Samples 160 Latest Sample 93 Average 266 Deviation 177

Figura 28.: Teste de escalabilidade da aplicação - Listagem de um jogo

Na figura 28 é possível validar que todos os utilizadores conseguiram processar o seu pedido em condições, sendo que ainda é apresentado o tempo médio para interpretar o pedido realizado pelo cliente. O tempo médio de resposta por parte do servidor é de 266 milissegundos.

4.5.5 *Outras técnicas*

Foram ainda utilizadas mais duas técnicas para garantir o melhor desempenho da aplicação. Uma delas foi configurações no servidor, de modo a garantir que determinados ficheiros fossem guardados na *cache*, mais concretamente ficheiros *JavaScript*, CSS, imagens entre outros. É necessário salientar que esta técnica só deve ser utilizada em ficheiros que sejam estáticos, ou então terá que ser definido um tempo de vida mais curto, garantindo-se que o ficheiro esta actualizado.

Outra técnica foi o *minify*, que permite reduzir o tamanho dos ficheiros CSS e *JavaScript*, facilitando assim a sua leitura, o que proporciona uma maior rapidez à aplicação quando necessita dos mesmos.

4.6 SISTEMA DE CONTROLO DE VERSÕES

O sistema de controlo de versões tem como função garantir o controlo de várias versões da aplicação. Neste produto foi utilizado o *git*, uma técnica que é muito utilizada quando existe um equipa a desenvolver um produto, ou seja, quando existe uma versão Pai, que é de onde derivam sempre as suas versões/*branchs* filhas. Normalmente é utilizado o nome *master* para a versão presente no servidor, e *develop* para o nome da versão dos programadores, sendo que a *develop* é filha da master. É uma boa prática que cada programador crie uma *branch* filha da *develop*, associada à funcionalidade que vai ser desenvolvida.

Posteriormente ao desenvolvimento da funcionalidade, essas alterações terão de ser enviadas para *develop*, de modo a serem integradas. Seguidamente é possivel verificar alguns dos comandos[27] utilizados:

- ***git pull*:** Permite actualizar todas as alterações que determinada *branch* sofreu;
- ***git push*:** Responsável por enviar todas as alterações para a *branch*;
- ***git add* nome_ficheiro:** Permite adicionar um ficheiro alterado/novo à *branch* em questão;
- ***git commit*:** Responsável por enviar todas as alterações para a *branch*;
- ***git checkout*:** Comando utilizado para descartas alterações nos ficheiros alterados;
- ***git checkout -b* nome_*branch*:** Cria uma nova *branch* a partir de uma *branch* especìfica;
- ***git merge*:** Responsável por resolver possíveis conflitos em determinados ficheiros;
- ***git status*:** Responsável verificar os ficheiros que foram alterados numa respectiva *branch*;

Em suma, esta abordagem permite que os programadores estejam a trabalhar no produto sem interferir com o trabalho dos outros. Essas modificações serão integradas nas *branch* principal de desenvolvimento (*develop*), podendo contudo existir conflitos na integração das *branchs* dos vários programadores, que depois são resolvidas (*git merge*) pelos programadores.

4.7 TESTES

Ao fim de implementar as funcionalidades na aplicação, é necessário realizar testes para garantir que o resultado corresponde ao que foi perspectivado nos requisitos e posteriormente nos diagramas concebidos. Estes testes podem ser realizados seguindo determinados passos, de modo a seguir o *workflow* da funcionalidade e assim verificar-se o resultado da operação.

Outra forma que pode ser utilizada é os testes unitários, que permitem testar a funcionalidade através do código fonte. Seguidamente vai ser abordada a ferramenta utilizada para testar aplicação a nível de testes unitários.

Foram ainda realizados testes de aceitação, que permitem testar funcionalmente a aplicação, ou seja, realizar operações na aplicação como se trata-se de um utilizador final. Este teste permite verificar o comportamento da aplicação.

O *PHPUnit*[28] é uma *framework* de testes unitários, que permite criar testes unitários em produtos que utilizem o PHP. Estes testes não costumam ser muito complexos, ou seja, é responsável por testar pequenos módulos de código. No exemplo seguinte é apresentado um dos testes realizado para aplicação. A figura 29 demonstram alguns testes unitários criados.

```php
class PlayerTest extends \UnitTestCase
{

    /**
     * Função responsável por testar a criação de um jogador
     */
    public function testNewPlayer()
    {
        $player                 = new Player();
        $player->player_id      = "1";
        $player->player_name    = "Henrique";
        $player->player_number  = "1";
        $player->player_position = "GR";
        $this->assertEquals($player->save(), true);
    }

    /**
     * Função responsável por testar a eliminação de um jogador
     */
    public function testDeletePlayer()
    {
        $player = new Player();
        $player = $player->getPlayer(1);
        $this->assertEquals($player->delete(), true);
    }

    /**
     * Função responsável por validar que só procura jogador com valores inteiros
     */
    public function testSelectPlayer()
    {
        $player     = new Player();
        $playerlist = $player->getPlayer("error");
        $this->assertEquals($playerlist, false);
    }
}
```

Figura 29.: Testes unitários referentes ao jogador

Os testes constantes na figura 29 representam os seguintes cenários:

- O primeiro teste é referente à criação de um jogador, sendo que para que o teste passe com sucesso, a função de inserir na base de dados, terá de devolve o valor *true*;
- O segundo teste é referente à remoção de um determinado jogador. Em primeiro lugar é necessário verificar se o mesmo existe, sendo seguidamente validado se o jogador foi eliminado ou não com sucesso;
- A terceira situação é para testar, que apenas apresenta a informação de um jogador, caso seja passado um valor inteiro. Existe uma validação na parte da função *getPlayer()* que valida se o *id* passado é um inteiro, caso contrário devolve vazio;

Depois de serem escritos os testes é necessário efectuar a sua validação, de forma a garantir as funcionalidades concebidas para o produto. Na ilustração (Fig.30) é possível analisar a realização dos testes.

```
C:\xampp\htdocs\apptese\tests>c:\xampp\htdocs\apptese\vendor\bin\phpunit
PHPUnit 5.7.21 by Sebastian Bergmann and contributors.

Runtime:       PHP 7.0.9
Configuration: C:\xampp\htdocs\apptese\tests\phpunit.xml

...                                                                 3 / 3 (100%)

Time: 2.94 seconds, Memory: 4.00MB

OK (3 tests, 3 assertions)
```

Figura 30.: Testes unitários validação

Na figura 30 é possível verificar a informação referente aos testes descritos, sendo que o mais importante é o que está sublinhado amarelo, onde clarifica que os testes correram com sucesso, garantindo assim as funcionalidades desenvolvidas para o produto.

Em suma, este passo é extremamente importante em qualquer aplicação, pois não é aconselhável disponibilizar um produto com erros, o que gera uma desconfiança por parte dos utilizadores finais. Os testes unitários servem para testar a funcionalidade de determinada parte do projecto, criando cenários de teste. Quanto aos testes de aceitação, referem-se a uma simulação no produto.

4.8 DEPLOYMENT

Este último passo é responsável por garantir o acesso da aplicação aos seus utilizadores. Para que aplicação seja corrida sem qualquer problema é necessário garantir que o servidor onde vai ser alojada tem requisitos para tal.

Qualquer aplicação tem as suas dependências, quer de bibliotecas ou mesmo de serviços. No PHP é utilizado o *composer* que permite guardar todas as dependências do projecto. Essas dependências são guardadas num ficheiro *composer.json*, que como o nome sugere é um ficheiro do tipo JSON. Assim sempre que for realizado um *deploy* para um servidor é preciso correr sempre o *composer* para instalar as dependências que serão guardadas na pasta *vendor*, que será sempre carregada por um ficheiro *autoloader*, de modo a ser utilizado no projecto sem qualquer problema.

4.8.1 *Alojamento*

Em relação ao alojamento da aplicação, foi utilizado o *heroku*[10], para que seja possível qualquer utilizador aceder à aplicação[11] via *browser*, como é pedido num requisito não funcional.

A escolha recaiu no *heroku* por ser possível obter um servidor grátis e que suporta a *framework* que foi utilizada na aplicação. No entanto existem diversas soluções para as aplicações PHP.

O servidor utilizado é limitado em recursos, tendo apenas 512 *MB* de *RAM*, contudo agora é suficiente, para demonstrar o que foi concebido. Posteriormente, será necessário migrar para um servidor mais robusto, de modo a conseguirmos obter o total controlo sobre o mesmo. Este servidor apenas permite que seja colocada a aplicação, ou seja, não permite configurar o servidor à nossa maneira, por isso é que não foram implementadas algumas técnicas como o *load balancer* entre outras.

10 Página web: `https://www.heroku.com/`
11 Url da aplicação : `https://teseum.herokuapp.com/`

5

CASO DE ESTUDO / APLICAÇÃO

Neste capítulo serão apresentadas as diversas funcionalidades implementadas na aplicação, permitindo verificar o produto em si, e como estão implementadas as soluções que foram descritas anteriormente. Será ainda abordada a API, com as respectivas chamadas.

Vão ser apresentados os resultados desenvolvidos, para dar resposta aos requisitos que o cliente solicitou, sendo que no decorrer da utilização e testes da aplicação foram surgindo novas ideias, e novas perceções em relação à potencialidade desta aplicação.

5.1 APLICAÇÃO - RESULTADOS

Nesta secção serão apresentados os módulos que foram desenvolvidos para a aplicação. Vai ser ainda exposta a estrutura da aplicação, de modo a permitir mostrar que estão a ser seguidas as normas do MVC.

Na figura 31 é possível verificar a estrutura da aplicação.

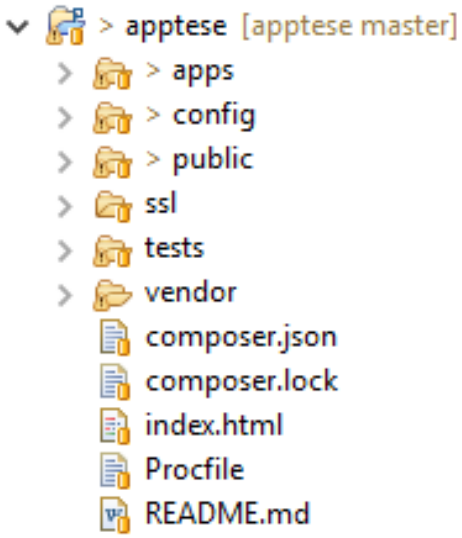


Figura 31.: Estrutura da aplicação

Na pasta *apps* está presente o *Backend*, *Frontend*, a API e as *migrations*. No caso do *backend* e *frontend*, cada módulo tem as suas configurações, mais concretamente a base de dados, podendo aceder a base de dados diferentes. No que respeita às configurações têm um ficheiro *module.php* que permite identificar o módulo em questão. Tem ainda presente os ficheiros responsáveis pelos *controllers*, *models* e *views* de cada módulo. Em relação à API tem, os ficheiros referentes às chamadas disponibilizadas. Por fim, temos a pasta *migrations* que contem as migrações referentes à base de dados, ou seja, sempre que exista uma alteração na base de dados, deve ser criada uma migração para que depois seja aplicada, por exemplo, na base de dados de produção.

No que respeita à pasta *config*, estão presentes todas as configurações do projecto em si, nomeadamente os serviços que a aplicação utiliza, como o *dependency injection*[1], as rotas da aplicação que vão ser utilizadas para comunicar com o servidor HTTP, de forma a que a aplicação identifique quem são os controladores e as respectivas acções a ser chamadas para dar resposta ao servidor HTTP. Tem ainda presente a configuração de cada módulo.

A pasta *public* é responsável por guardar todos os ficheiros *javaScript*, CSS, *plugins* entre outros. A pasta SSL contem o certificado SSL da aplicação. O directório *tests* contem os testes unitários realizados para testar a aplicação, bem como, as suas configurações para que fosse possível correr os testes.

Por fim, temos a pasta *vendor*, que é responsável por guardar as instalações das dependências presentes no ficheiro *composer.json*. O *composer.lock* guarda as informações sobre as dependências que foram instaladas. Dentro da pasta *vendor* encontra-se o ficheiro *autoload.php* que permite que a aplicação carregue todas as dependências instaladas.

5.1.1 *Backend*

Este módulo da aplicação é responsável por todo o conteúdo apresentando naquela. Neste tópico serão apresentadas algumas funcionalidades deste módulo da aplicação, sendo que este módulo destina-se apenas aos administradores da aplicação, com o intuito de aumentar a segurança e confiabilidade de toda a informação introduzida na aplicação.

5.1.1.1 *Autenticação e perfil administradores*

A figura 32 representa a funcionalidade de acesso ao *backend*. Para realizar qualquer operação no *backend*, é necessário ter conta de administrador. Esta medida de segurança permite que toda a informação apresentada na aplicação seja controlada pelo administrador.

1 permite guardar várias dependências numa classe, para utilizar em todo o projecto

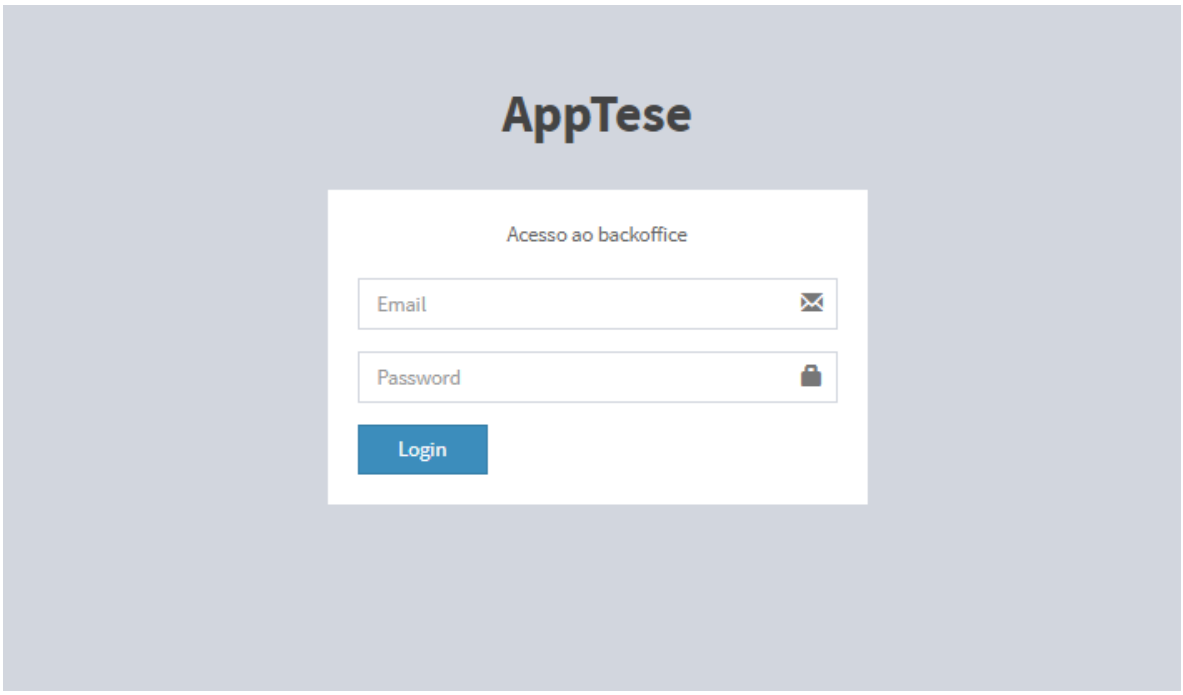


Figura 32.: *Backend - login*

Na figura 33 é possível verificar a opção de editar o perfil do administrador, de modo a poder alterar a sua informação caso seja necessário.

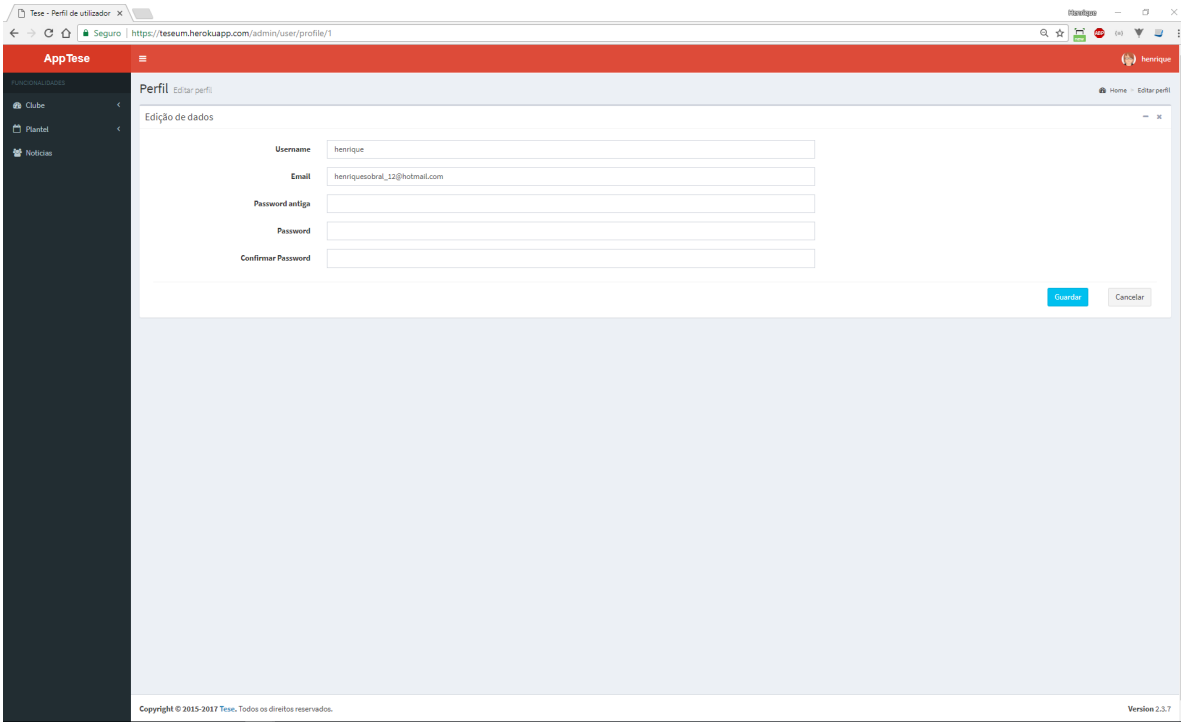


Figura 33.: *Backend* - Editar perfil

Como todas as aplicações que têm autenticação, também existe a funcionalidade do administrador fazer *logout*, de modo a terminar a sessão iniciada na aplicação.

5.1.1.2 *Gestão de histórias do clube*

Na figura 34 é possível verificar toda a informação associada à história do clube. Cada registo tem um *link*, de forma a ser realizada quer a edição da história quer a eliminação da respectiva história. O administrador dispõe ainda de um botão antes da listagem, o que permite a criação de uma nova história.

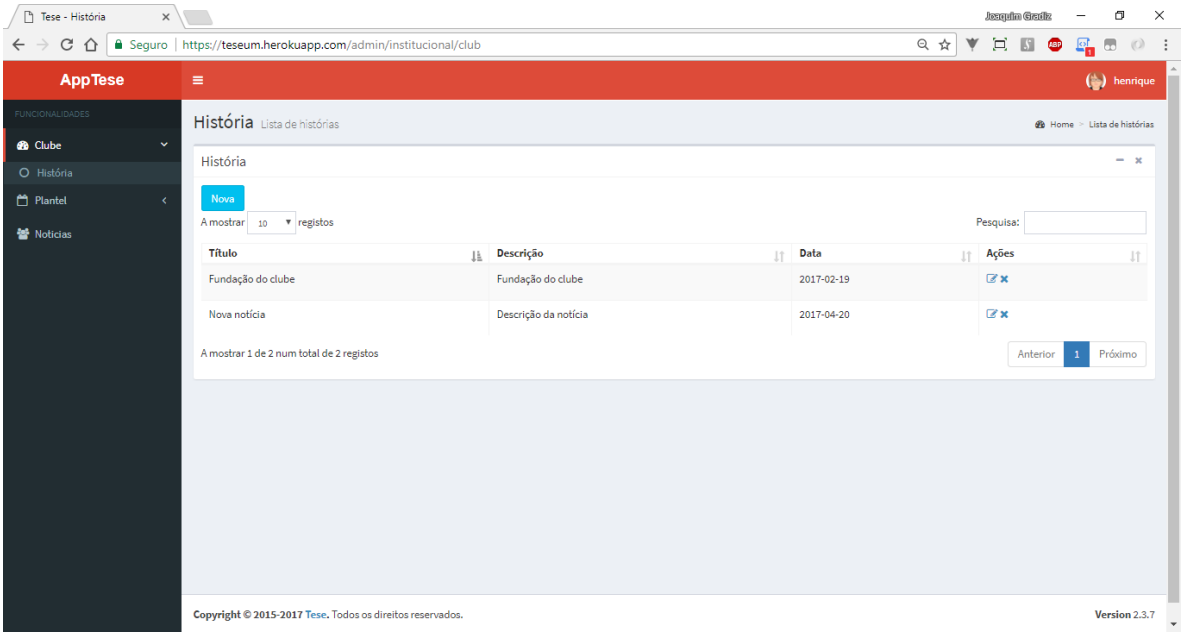


Figura 34.: *Backend* - Listagem de histórias do clube

Para realizar a criação de uma nova história o administrador tem de clicar no botão anteriormente mencionado para conseguir efectuar a operação. Na figura 35 é possível visualizar o formulário responsável por criar a história.

Importa salientar que o mesmo formulário é utilizado para editar a história, tendo os campos associados à mesma, já preenchidos com a informação da história a editar. Contudo, é necessário referenciar, que todos os formulários têm uma validação feita pelo *jQuery Validation*, como é possível verificar na figura 35.

Ao fim de submeter o formulário o utilizador será reencaminhado para a listagem de histórias, surgindo uma mensagem com a informação de sucesso ou erro, de modo a informar o administrador sobre o estado da operação.

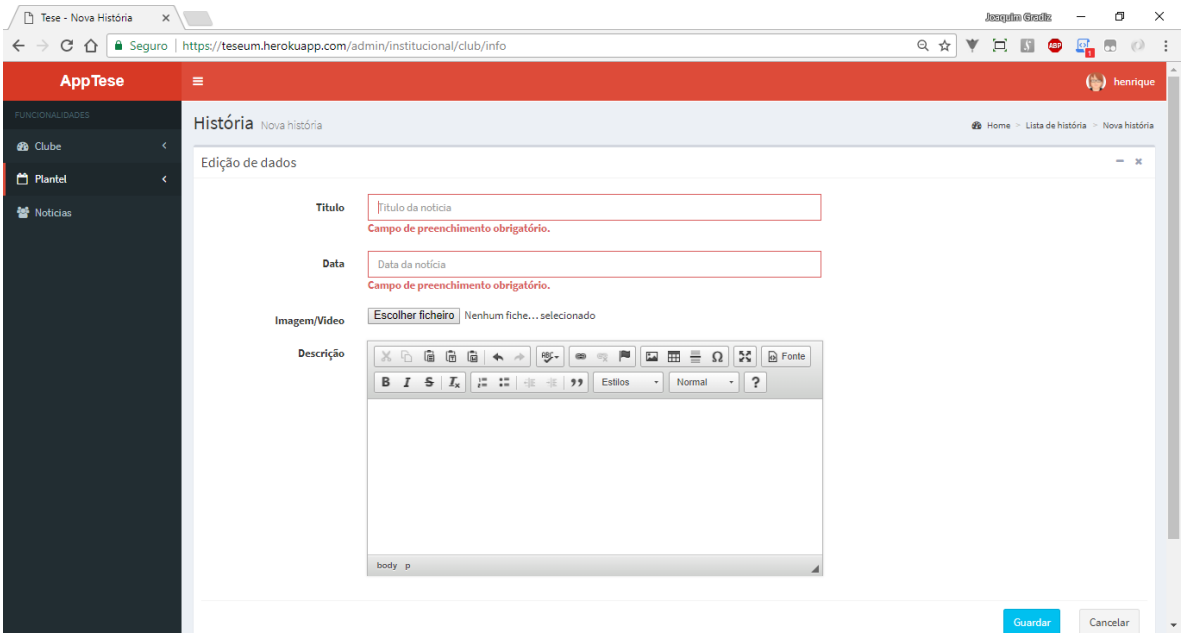


Figura 35.: *Backend* - Adicionar história

No que respeita à eliminação, é necessário clicar sobre a funcionalidade de eliminar e seguidamente irá aparecer um *modal*[2] como o da figura 36, para que o administrador confirme se pretende eliminar a respectiva história.

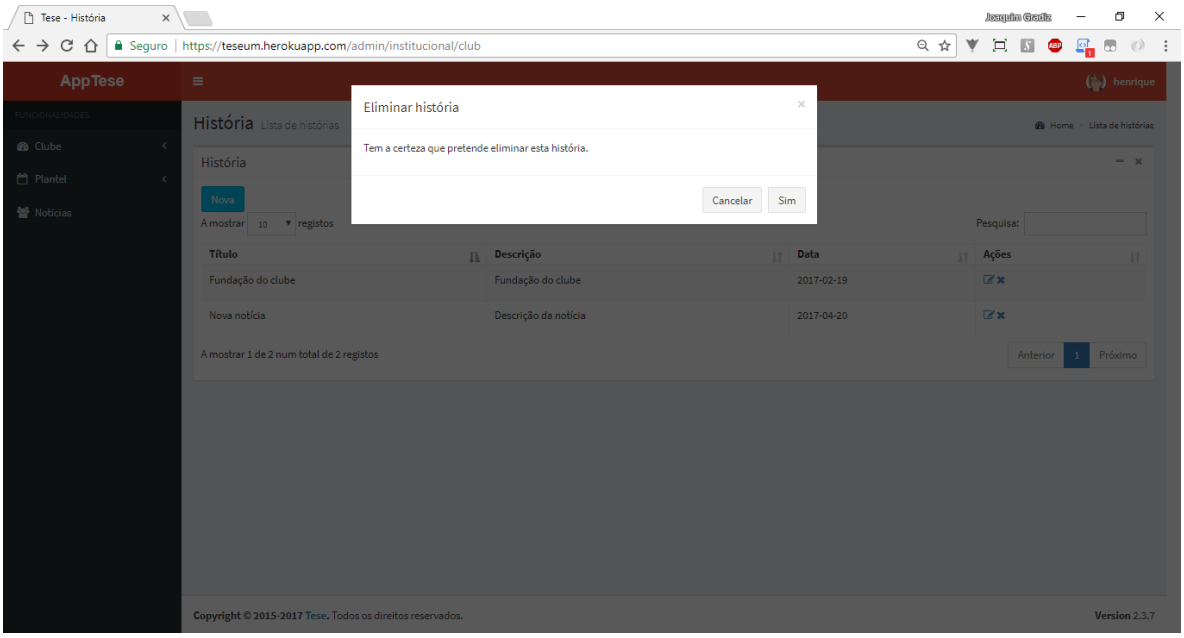


Figura 36.: *Backend* - Remover história

2 Página web: https://getbootstrap.com/docs/3.3/javascript/#modals

5.1.1.3 *Gestão de equipa*

A gestão da equipa segue o mesmo processo de *workflow* que as histórias e como todo o *backend*. No entanto a gestão de equipa engloba a edição de jogadores, treinadores e das respectivas épocas.

No caso dos jogadores, é possível inserir informações, como o seu nome, posição, número e foto, informações essas interessantes para que posteriormente seja mais fácil para o utilizador saber quais os jogos que àquele jogador disputou.

Em relação aos treinadores, também é possível inserir o seu nome ou foto, existindo ainda um campo para marcar se o treinador está activo ou não, podendo assim, na listagem da equipa, diferenciar-se do treinador actual.

Em relação às épocas, esta funcionalidade permite criar a época a que serão associados os jogos, o que permitirá fazer o cálculo de todas as estatísticas dos jogos. É necessário referenciar que foi adicionado uma *checkbox* que possibilita identificar a época que está a decorrer.

5.1.1.4 *Gestão de jogos*

A gestão dos jogos vai ser fundamental para registar toda a informação associada ao jogo, bem como, os intervenientes, eventos do jogo entre outros. Nesta parte é possível gerir a informação sobre o estádio onde serão introduzidos determinados detalhes, como o nome do estádio e a sua capacidade. O jogo também tem sempre um árbitro associado, algo que também teve de ser contemplado, de modo a fornecer a gestão dos árbitros.

Foi também criada uma funcionalidade que permite gerir as competições em que determinado clube participa, sendo também adicionada a gestão de equipas adversárias e dos eventos associados ao jogo. Estas opções foram concebidas, com o intuito de possibilitar a adaptação a qualquer clube porque nem todos participam nas mesmas competições. É necessário expor que em relação a gestão de eventos, possibilita que a aplicação se adapte a qualquer desporto. Com estas abordagens é possível oferecer agilidade à aplicação desenvolvida.

Em relação à gestão da informação referente aos jogos, serão guardados todos os dados dos jogos. Na figura 37 é possível verificar o formulário que deve ser preenchido para colocar a informação referente ao jogo. A informação vai ser constituída pela época a que se refere o jogo, em que competição se enquadra o jogo, onde o jogo é realizado, entre outras, sendo necessário referir que é possível guardar o resultado do jogo, bem como, o número de adeptos presente no jogo.

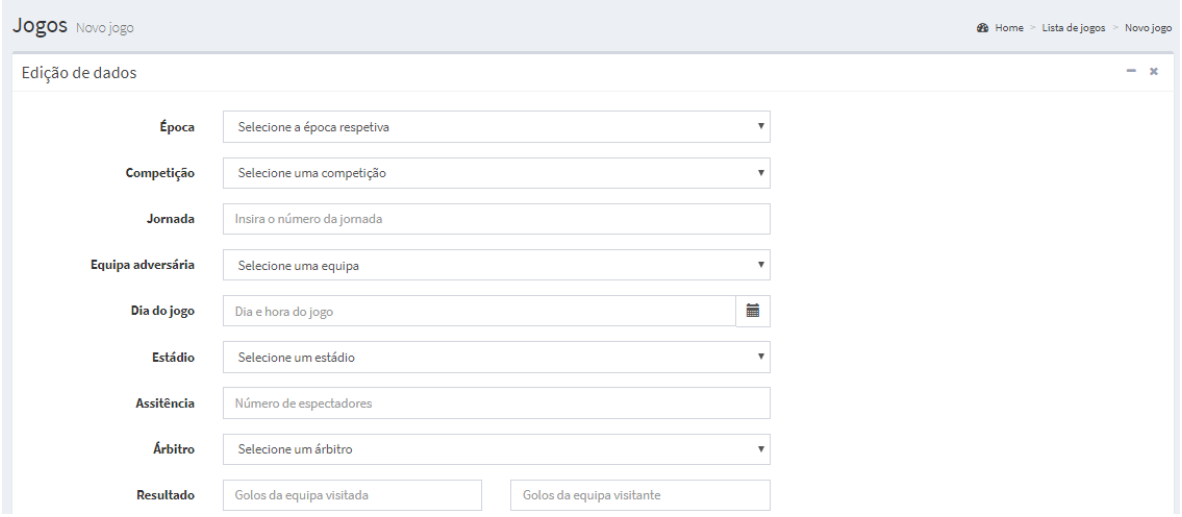


Figura 37.: *Backend* - Gestão de jogos (Informações gerais)

A figura 38 permite verificar como são associadas as informações complementares referentes ao jogo. Existe ainda uma listagem dos jogadores onde serão seleccionados os titulares e suplentes, e onde será também necessário associar o treinador. O jogo tem associados eventos, eventos esses inseridos neste formulário, onde o administrador dispõe de botões para ir adicionando eventos, visto não existir um número certos de eventos a decorrer em determinado jogo. O administrador em relação aos eventos apenas precisa de associar o jogador a um determinado evento, registando o minuto do mesmo. No entanto, caso seja substituição serão associados dois jogadores.

Estas informações são fundamentais para o que vai aparecer no *frontend*, pois é com base nesta informação que vai ser possível construir as estatísticas referentes aos jogadores, treinadores e equipa em geral. Com base nestas estatísticas, vai ser possível também fazer a análise de confrontos entre equipas e jogadores.

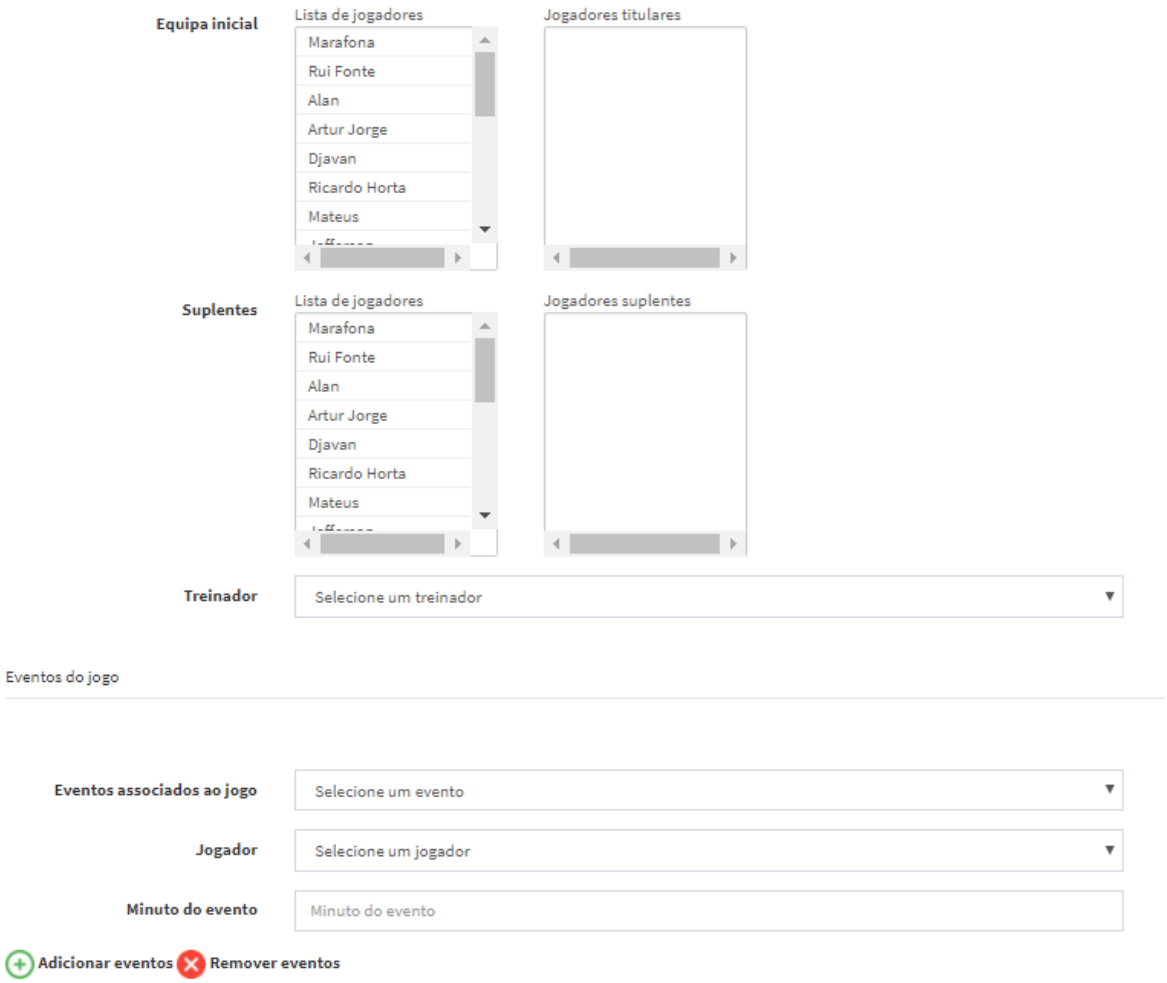


Figura 38.: *Backend* - Gestão de jogos (Jogadores, treinadores e eventos)

A gestão de jogos ainda tem associada a gestão das estatísticas de um determinado jogo, como é possível validar na figura 39, onde cada equipa terá as suas estatísticas.

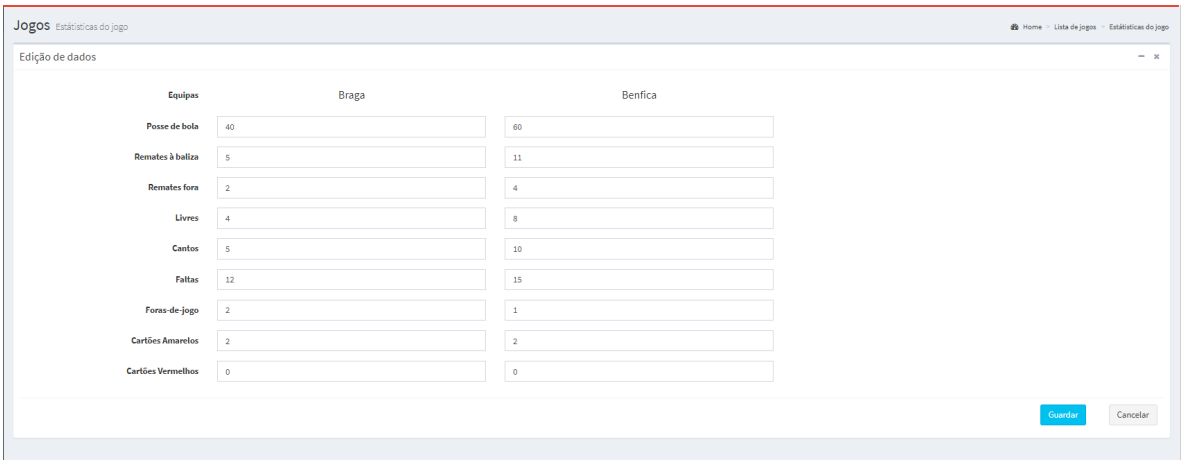


Figura 39.: *Backend* - Estatísticas do jogo

5.1.1.5 *Gestão de notícias do clube*

Esta parte da aplicação vai permitir gerir as notícias associadas ao clube, contendo a informação como imagens, título, descrição e a data da sua publicação. Optou-se por incluir a data da notícia, permitindo aos utilizador verificar em que dia a notícia foi inserida na aplicação.

5.1.2 *Frontend*

O *frontend* é o módulo responsável por apresentar todas as informações aos utilizadores, sendo também o responsável por interagir com os utilizadores. Nesta parte da aplicação vai ser possível verificar toda a informação do clube, mais concretamente jogos, notícias, jogadores, equipa, entre outros.

5.1.2.1 *Utilizador*

No que respeita ao utilizador, foi implementando um sistema de *login*, já a pensar em futuras funcionalidades que venham a ser desenvolvidas, para permitir que o mesmo consiga interagir mais com a aplicação. Essas possíveis funcionalidades serão descritas no último capítulo.

A figura 40 mostra o formulário de *login* para aceder à aplicação. No caso de o utilizador conseguir realizar *login* com sucesso, irá receber uma mensagem com informação de sucesso. O utilizador pode registar-se na aplicação, depois de realizar essa operação vai receber um *email* a dizer que foi registado com sucesso.

O utilizador tem ainda a possibilidade de realizar *login* através do *facebook*, podendo assim utilizar uma conta das redes sociais.

Quando é implementado um sistema de *login*, é necessário conceber a opção de poder recuperar a *password* e posteriormente é enviado um *email* para o utilizador com a nova *password*.

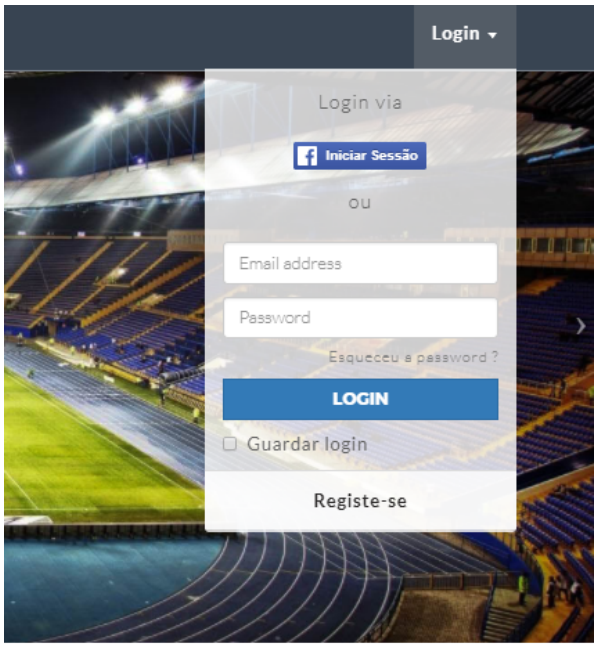


Figura 40.: *Frontend - login*

Existe ainda há possibilidade do utilizador poder alterar os seus dados de perfil, como *username*, *email* e *password*. Para alterar *password* é necessário que o utilizador insira sempre a *password* antiga. No entanto quando for realizada a alteração à *password* é preciso confirmar a nova *password*.

5.1.2.2 *História*

A história permite aos utilizadores consultar a informação sobre o clube, acontecimentos históricos que ocorreram ao longo da vida do clube, bem como, conquistas, entre outros acontecimentos.

Figura 41.: *Frontend* - Listagem de histórias

Na figura 41 é possível, verificar a exposição sobre a informação das histórias associadas ao clube. Optou-se por utilizar este tipo de cronologia, permitindo ao utilizador visualizar os eventos consoante o tempo, associando assim os acontecimentos como se fosse uma cronologia.

5.1.2.3 *Notícias*

As notícias permitem aos utilizadores estar a par de todas as notícias referentes ao clube, permitindo assim que os seus adeptos estejam a par de tudo o que se passa no seu clube.

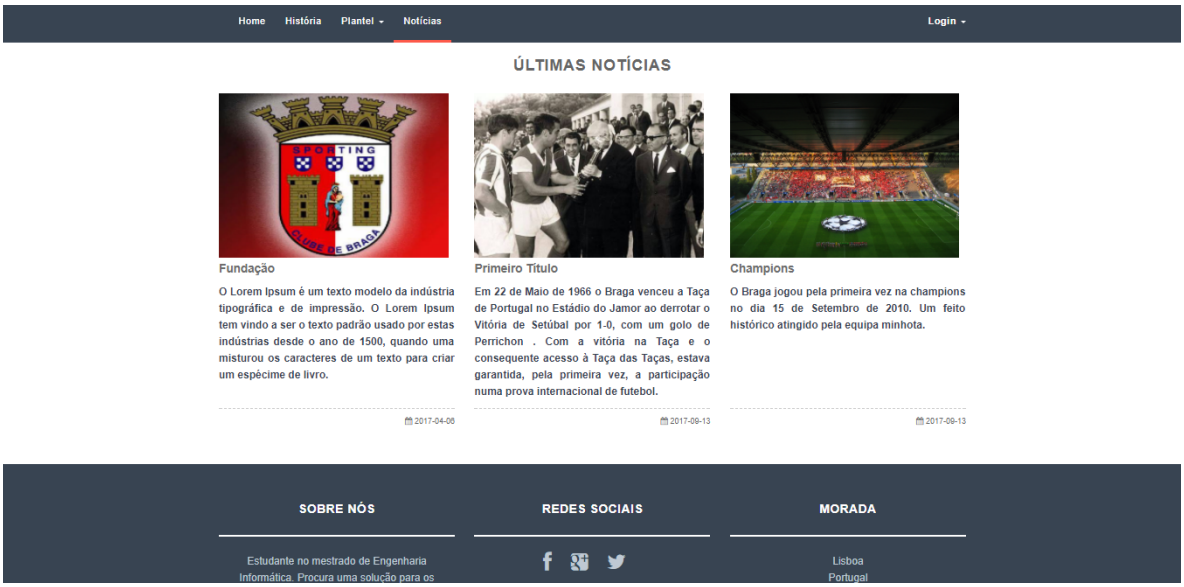


Figura 42.: *Frontend* - Listagem de notícias

A figura 42 permite verificar como é que as notícias são apresentadas aos utilizadores. É possível verificar que apenas será apresentada uma parte da notícia, sendo que ao clicar na notícia será reencaminhado para a página com a informação completa sobre a mesma.

5.1.2.4 *Equipa*

No que respeita à equipa, optou-se por mostrar todos os jogadores associados a uma determinada época, ou seja, só aparecem os jogadores que realizaram jogos na respectiva época. No início da página existe uma *combobox* que contem as épocas que estão na aplicação. Sempre que a *combobox* for alterada, vai ser realizado um pedido AJAX para carregar a informação de determinada época.

A figura 43 permite visualizar um resumo de todos os jogos que foram realizados em determinada época, organizados por competição. Seguidamente aparece a listagem dos últimos cinco jogos de cada época, no caso de o utilizador pretender ver todos os jogos, terá essa opção, sendo reencaminhado para a página que apresenta todos os jogos de uma respectiva época.

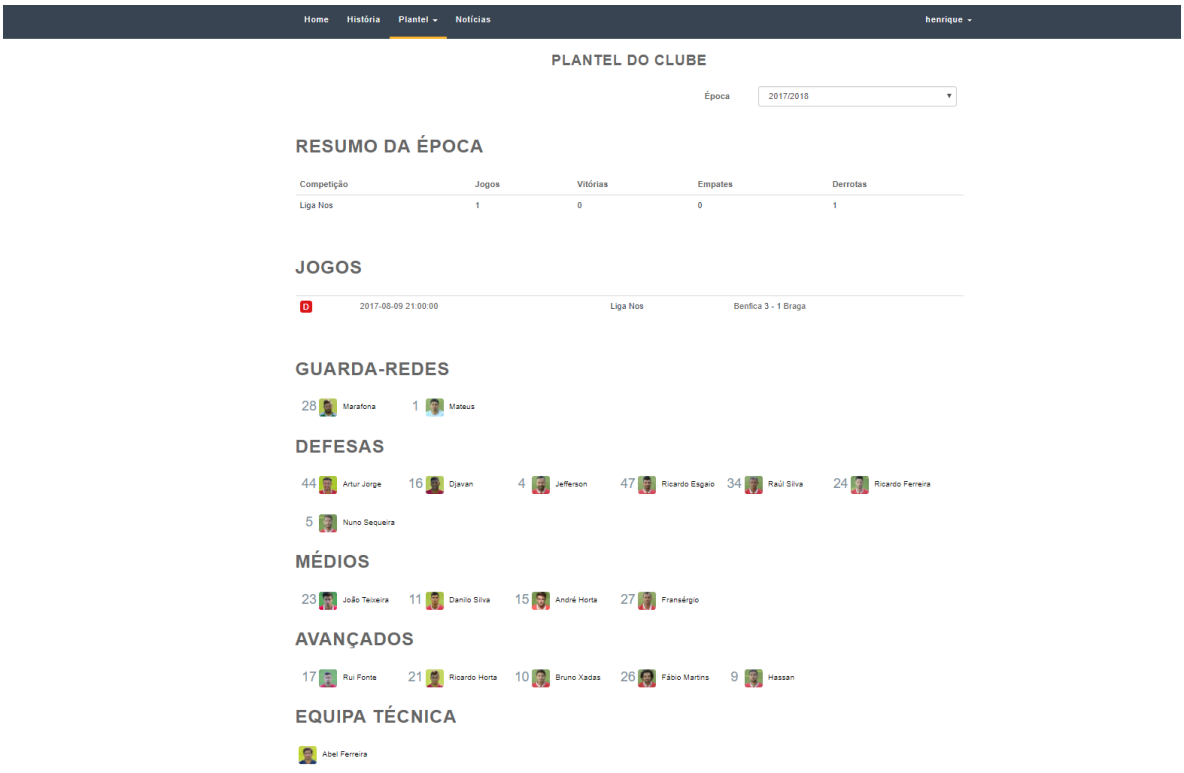


Figura 43.: *Frontend* - Lista dos elementos da equipa

Nesta funcionalidade é apresentado o treinador referente à época em questão, no entanto se houver vários treinadores numa época, os treinadores que já não fazem parte do plantel vão aparecer de uma forma não tão destacada.

5.1.2.5 *Jogos*

Em relação aos jogos, vai existir uma parte da aplicação que vai apenas apresentar todos os jogos que a equipa realizou, durante determinada época. Sendo necessário salientar que nesta funcionalidade também se dispõe de uma *combobox*, de modo a listar os jogos consoante a época escolhida.

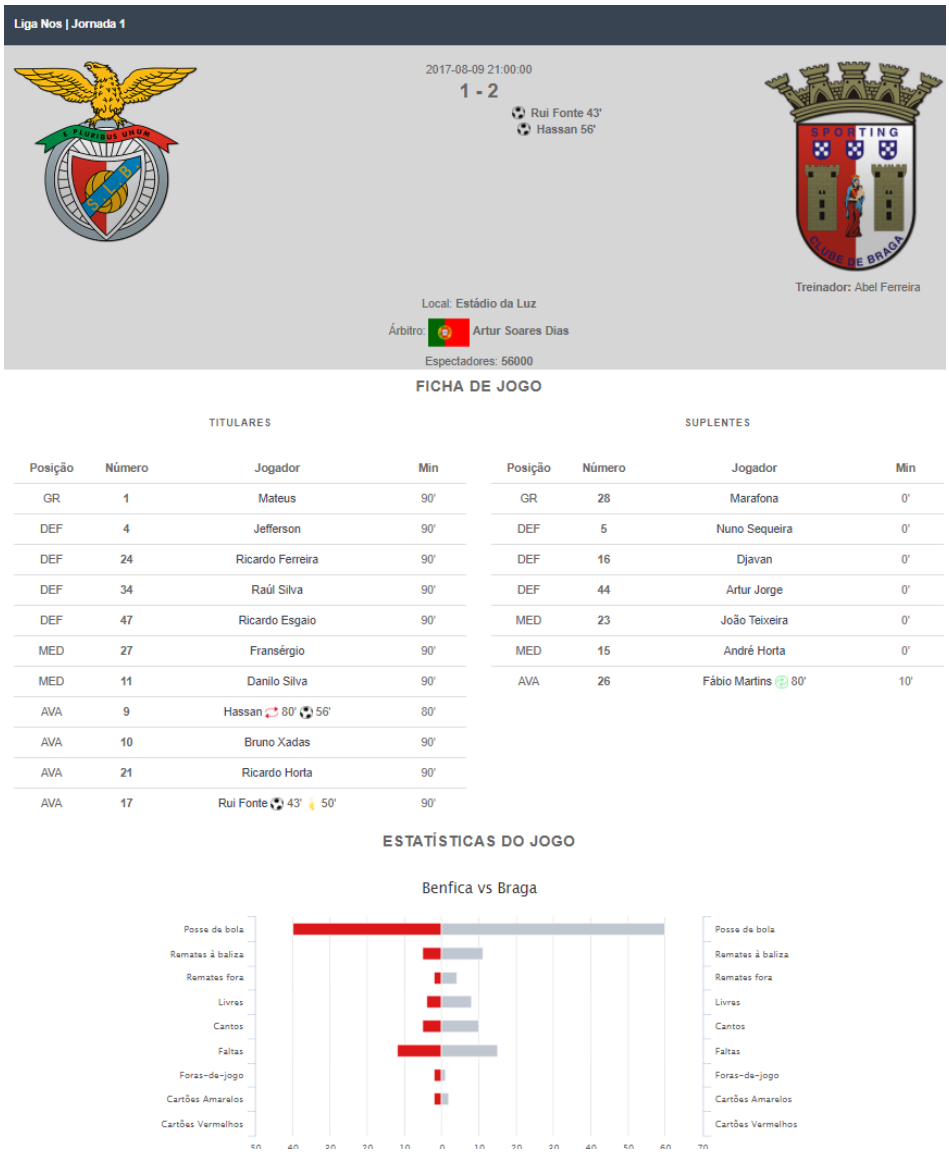


Figura 44.: *Frontend* - Informação do jogo

A figura 44 permite verificar a informação de um jogo, e para isso o utilizador terá de clicar no resultado do jogo que pretende visualizar. Como podemos ver na figura 44 no topo irá ser apresentada a informação dos clubes intervenientes, os marcadores dos golos, a equipa de arbitragem, o resultado e dados associados ao jogo como espectadores e competição a que o jogo pertence. Os dados do treinador só surgem no caso de ser o clube detentor da aplicação, bem como, os dados dos jogadores e marcadores de golos.

Posteriormente será exposta a informação referente aos jogadores que iniciaram o jogo e aos suplentes. Existe também a informação dos eventos que ocorreram durante o jogo.

No fim desta página será apresentado um gráfico onde estão presentes as informações sobre as estatísticas relacionadas com os jogos em questão, desde posse de bola, remates, entre outras informações.

5.1.2.6 *Confrontos - Análise Raio-X*

O utilizador tem a possibilidade de analisar o desempenho da sua equipa, tendo ao dispor duas *combobox* para escolher o adversário e a competição que pretende analisar, tal como representa a figura 45. No entanto, é necessário salientar que caso o utilizador não escolha nem uma equipa, nem uma competição, ele faz uma análise geral. Esta funcionalidade permite aos utilizadores saberem o desempenho da sua equipa consoante as equipas adversárias ou a competição em questão.

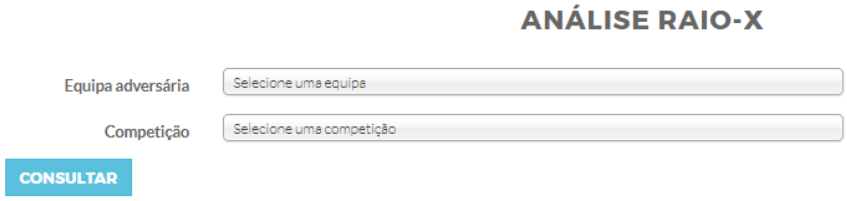


Figura 45.: *Frontend* - Análise Raio-X

A figura 46 apresenta informação de todos os jogos que foram disputados contra o Benfica. Primeiro surge a informação sobre os jogos em questão, oferecendo a possibilidade de o utilizador poder consultar a informação referente àquele.

Seguidamente (Fig.46) são apresentados dois gráficos circulares onde o primeiro representa o número de jogos que a equipa disputou, sendo os dados agrupados por vitória, empate ou derrota. O outro gráfico circular representa o número de golos marcados e sofridos nos confrontos entre as duas equipas.

Os gráficos de barras representam a mesma informação, no entanto aqui encontra-se separada pelos jogos realizado em casa ou fora, permitindo ao utilizador analisar onde é que a sua equipa é mais forte.

Esta funcionalidade também foi implementada, para os jogadores e treinadores, funcionando da mesma forma, comparando apenas o desempenho do jogador e treinador em vez de ser da equipa.

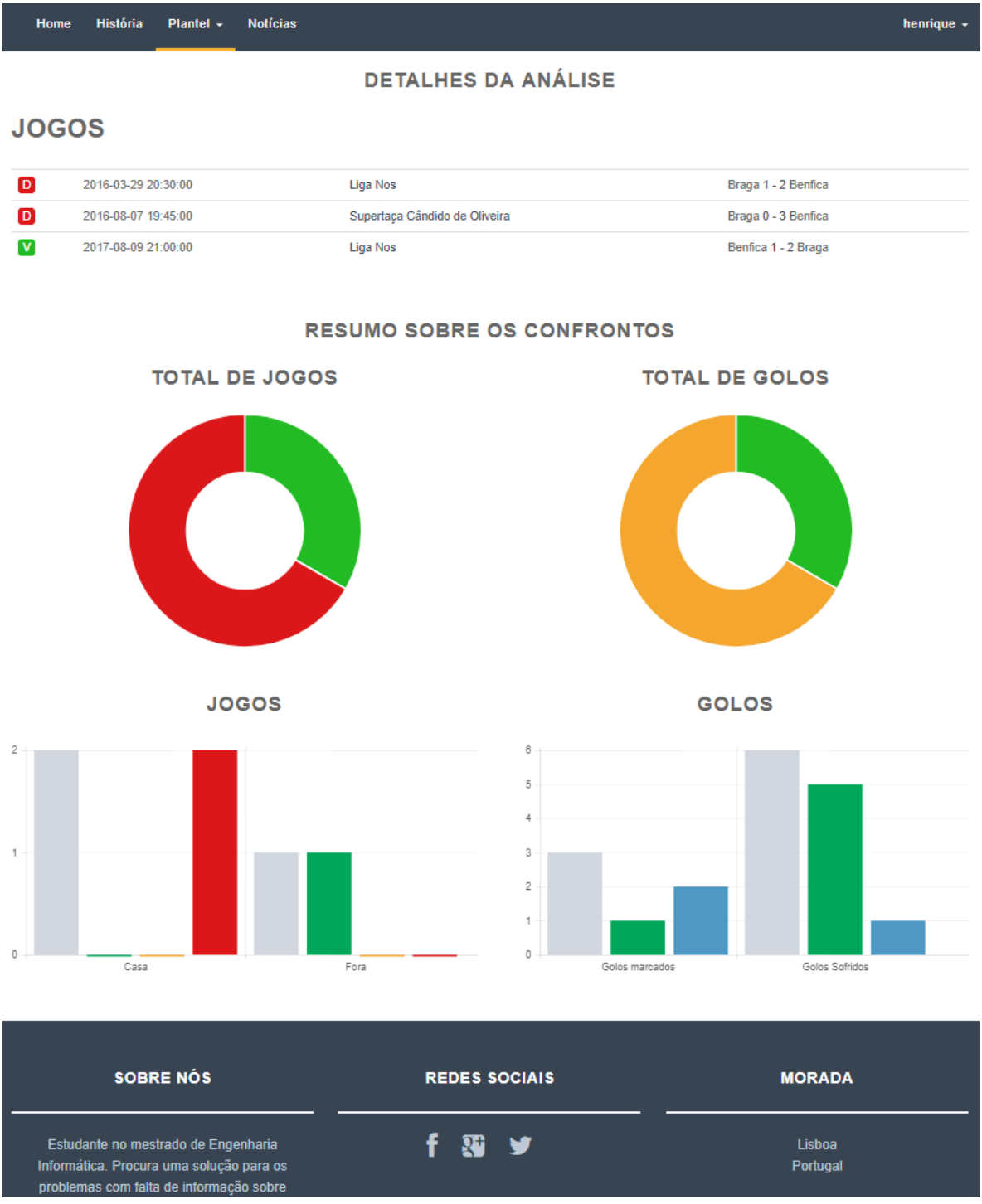


Figura 46.: *Frontend* - Análise Raio-X

5.1.2.7 *Jogador e treinador*

Os elementos das equipa têm uma funcionalidade que permite ao utilizador consultar a informação sobre o mesmo. A figura 47 representa a informação sobre um jogador, que é similar para o treinador. Existe na mesma a *combobox* com as épocas, possibilitando ao utilizador consultar informações consoante a época seleccionada.

Na primeira parte da página é apresentada a informação sobre o jogador, aparecendo posteriormente a informação sobre o resumo da época seleccionada. No resumo irão aparecer todas as informações sobre o número de jogos que fez em determinada competição.

Na figura seguinte (Fig.47) é possível verificar os últimos jogos disputados pelo jogador, bem como, o histórico do jogador desde que entrou no clube. Por fim, existem dados sobre curiosidades do jogador e eventos a que o jogador está associado como golos, cartões amarelos entre outros. Importa referir que existe a opção de consultar confrontos de determinado jogador.

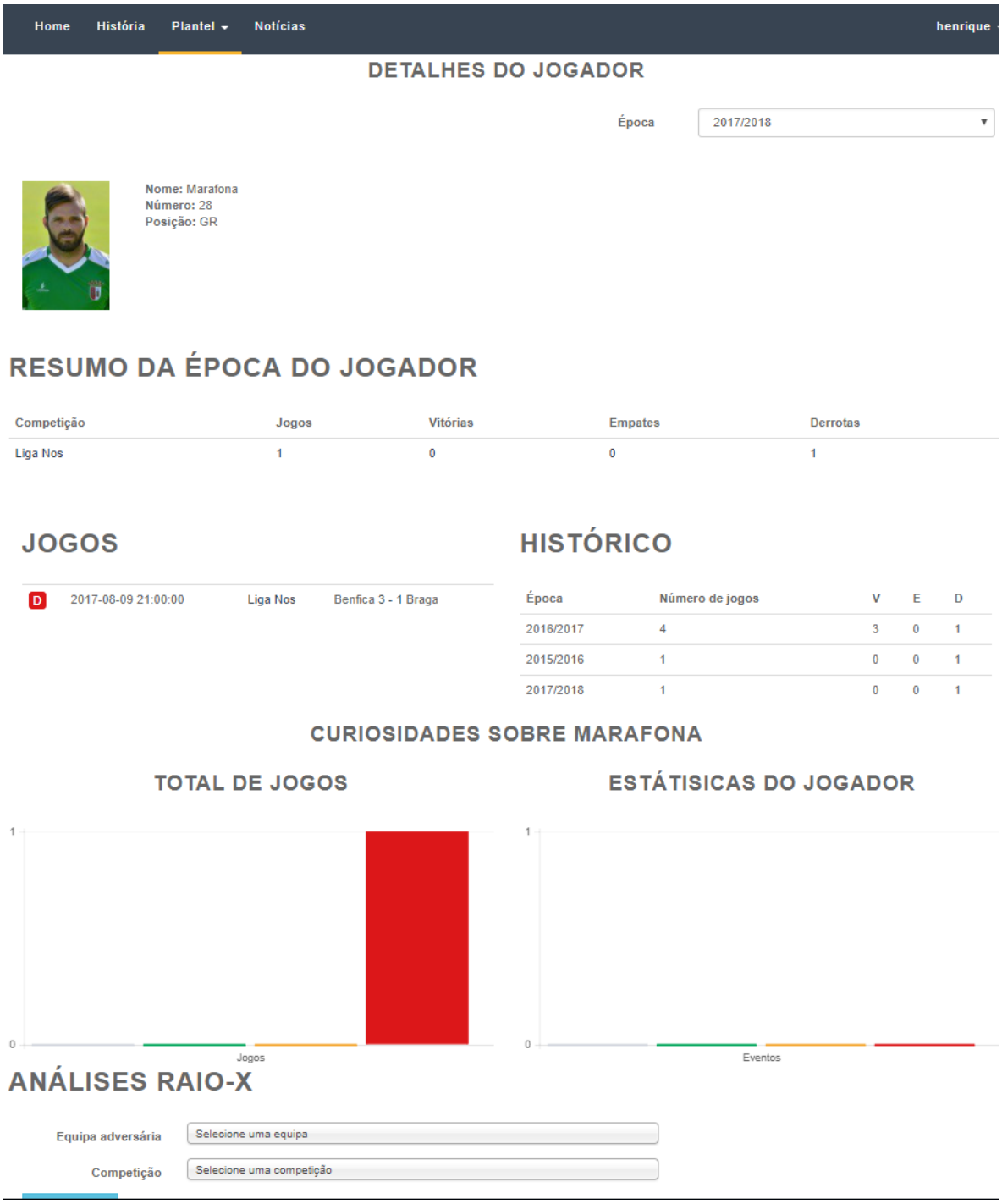


Figura 47.: *Frontend* - Informação do jogador

5.1.3 *API*

Nesta secção serão apresentados os *endpoints* que foram criados, de forma a garantir resposta caso seja necessário no futuro. Quando desenvolvida uma API, é fundamental fazer a documentação da mesma, onde ficam documentadas as chamadas que foram desenvolvidas. Esta abordagem vai permitir uma fácil integração noutras aplicações.

Os *endpoints* são *url's* que são definidos para que a API perceba que recursos tem que utilizar para processar o pedido à aplicação. No entanto, é necessário definir o método que vai ser utilizado. Caso o *url* corresponda mas o método não corresponda ao que está definido no *endpoint*, ocorrerá um erro da gama 400 visto ser um pedido mal formulado por parte do cliente.

Na seguinte tabela (Tab.1) serão enumeradas todos os *endpoints* concebidos:

URL	*Método*	*Parâmetros*	*Descrição*
https://teseum.herokuapp.com/api/v1/games	GET	seasonId, competitionId e gameId	Permite consultar a lista de todos os jogos
https://teseum.herokuapp.com/api/v1/players	GET	season e id	Devolve a lista de todos os jogadores
https://teseum.herokuapp.com/api/v1/seasons	GET	id	Devolve a lista de todos as épocas
https://teseum.herokuapp.com/api/v1/competitions	GET	id	Devolve a lista de todos as competições

Tabela 1.: Lista de *endpoints*

5.1.3.1 *Jogos*

No que respeita ao *endpoint* referente aos jogos, é responsável por devolver os jogos consoante o pedido do utilizador. O utilizador tem a possibilidade de definir o que pretende filtrar, podendo obter apenas a informação de um determinado jogo, ou de uma determinada época e competição. Esta abordagem permite obter apenas o jogo que pretende.

A figura 48 apresenta uma resposta da API com a informação de todos os jogos do clube.

```
// http://localhost/apptese/api/v1/games

[
  {
    "competition": "Taça de Portugal",
    "season": "2016/2017",
    "fixture": "Oitavos",
    "date": "2017-03-22 18:00:00",
    "hometeam": "Braga",
    "awayteam": "Porto",
    "result": {
      "goalHome": "3",
      "goalAway": "1"
    }
  },
```

Figura 48.: API - *endpoint* jogos

A figura 48 representa a resposta que a API devolve. Cada jogo vai conter esta informação, onde é descrita a competição a que o jogo pertence, qual é época a que o jogo pertence entre outras informações. No caso do resultado, optou-se por esta estrutura de forma a estruturar melhor a informação.

5.1.3.2 *Jogador*

Em relação aos jogadores foi defenido um *endpoint* que possibilita o envio de informação para que seja possível listar os jogadores que representam o clube.

No *request* podem ser inseridos os parâmetros de pesquisa, de forma a filtrar os jogadores que pretende receber. É possível colocar o *id* da época (obtido no *endpoint* referente às épocas que pretende), e com isso são apresentados apenas os jogadores daquela época. No entanto, é ainda possível definir o *id* de um respectivo jogador, conseguindo apresentar a informação de determinado jogador.

```
// http://localhost/apptese/api/v1/players?season=3

[
  {
    "id": "2",
    "name": "Marafona",
    "position": "GR",
    "number": "28"
  },
  {
    "id": "4",
    "name": "Rui Fonte",
    "position": "AVA",
    "number": "17"
  },
  {
    "id": "24",
    "name": "Artur Jorge",
    "position": "DEF",
    "number": "44"
  },
```

Figura 49.: API - *endpoint* jogadores

A figura 49 apresenta a forma como a API responde ao pedido que foi realizado, no entanto, é preciso verificar que foi enviado o parâmetro *season*, que permite apenas ver os jogadores que jogaram na época com *id* 3 (referente a 2017/2018).

5.1.3.3 *Competições e Épocas*

A construção destes *endpoints* foi importante para que se perceba quais são os *id's* das competições ou épocas que vão ser utilizados nos filtros dos jogos e dos jogadores.

A figura 50 apresenta a resposta associada ao *endpoint* das competições. O *endpoint* referente as épocas terá a mesma estrutura apenas alterando os valores da resposta.

```
// 20170923202923
// http://localhost/apptese/api/v1/competitions

[
  {
    "id": "1",
    "name": "Liga Nos"
  },
  {
    "id": "2",
    "name": "Taça de Portugal"
  },
```


Figura 50.: API - *endpoint* competições/épocas

A implementação da API foi realizada, de modo a preparar aplicação para futuras funcionalidades da mesma, quer para uma possível aplicação móvel ou para comunicação com outras aplicações.

No entanto é preciso referenciar que ainda não foi incluída a incorporação de uma API-key, mas é algo que terá de ser levado em conta de forma a gerir os acessos à API. Em relação aos pedidos, existem métodos de bloquear endereços *IP's* que estejam a realizar demasiados pedidos, de forma a deixar o servidor HTTP em baixo. Essa medida pode ser efectuada através do servidor na gestão de *firewall*.

6

CONCLUSÃO E TRABALHO FUTURO

Neste capítulo serão abordados os desafios encontrados ao longo da dissertação, desafios esses que contribuíram para o nosso crescimento tecnológico. Serão ainda abordadas as funcionalidades e potencialidades da aplicação desenvolvida, visto que este *software* está no início da sua vida, e que no futuro ainda pode ser melhorado.

6.1 CONCLUSÕES

Esta dissertação teve como principal objectivo conceber uma aplicação *web* para dar resposta aos fãs de qualquer clube, oferecendo-lhes a possibilidade de consultar toda a informação relevante sobre o mesmo. No entanto, também existiu a necessidade de explorar e descrever de que forma se implementa uma aplicação, mais concretamente os passos a seguir para construir uma aplicação de forma coerente e robusta.

No mundo empresarial existem sempre diversas equipas responsáveis por várias fases do projecto, nesta dissertação, houve a necessidade do autor explorar todas as responsabilidades de cada uma das equipas, desde análise do estado da arte, passando pelo levantamento de requisitos, terminando na concepção e *deploy* da aplicação.

Na etapa inicial desta dissertação foram realizadas reuniões com o cliente, de forma a entender quais as suas pretensões. Com isso foi possível começar a detectar algumas funcionalidades que tinham de ser desenvolvidas para aplicação.

Posteriormente ao primeiro contacto com o cliente, começou-se a análise de requisitos, de modo a conhecer o modelo de domínio em que aplicação se insere e estudar as soluções já existentes no mercado, de forma a colmatar as suas falhas. Nesta fase também se começou a definir as tecnologias que melhor se adaptariam à solução a desenvolver, o que ajudou a melhorar conhecimentos, foram ainda consultadas diversas opiniões e lida diversa bibliografia.

A dissertação permitiu expandir os conhecimentos sobre as tecnologias relacionadas com as aplicações *web*, sendo de reforçada importância a aquisição de conhecimentos sólidos sobre esta área. Foram utilizadas diversas tecnologias que até aqui eram desconheci-

das, desde testes unitários (foi fundamental para testar toda aplicação a nível de código), implementação de *web services* e técnicas de *performance* em aplicações *web*.

No que respeita ao desenvolvimento da aplicação em si, no início existiram algumas adversidades, pois foi necessário aprender e explorar as funcionalidades da *framework phalconPHP*. A utilização do HMVC foi o primeiro desafio sentido, uma vez que era necessário estruturar a aplicação, dividindo a mesma em módulos e definir as configurações para cada um deles. Esta *framework* utiliza o OOP, tornando por isso necessário explorar este paradigma, com o intuito de alcançar um desenvolvimento mais fluente.

Em relação ao ORM da *framework* notou-se que em alguns casos, mais concretamente em relações com tabelas, o ORM realiza *querys* desnecessárias o que aumentava o *overhead* da aplicação. A solução encontrada foi realizar a *query* com o SQL puro, o que aumentou o desempenho da aplicação. Esta solução só foi explorada quando era utilizado a relação entre tabelas.

As aplicações cada vez mais têm que oferecer segurança aos seus utilizadores. Para tal, sentiu-se a necessidade de explorar falhas de segurança nas aplicações *web*, sendo por isso fundamental consultar o OWASP para verificar como proteger a aplicação. Foi explorado o SSL o que deixa sempre uma boa indicação aos utilizadores, uma vez que a comunicação entre o servidor e cliente fica sempre encriptada. Este estudo permitiu desenvolver um certificado SSL para ser utilizado na aplicação.

Como foi referido, tentou-se uma colaboração com o zerozero, uma das aplicações estudadas de forma a explorar o modelo de domínio, sem nunca se obter qualquer resposta. Esta colaboração seria fundamental para conseguir obter informações de determinado clube, facilitando o trabalho do administrador da aplicação, sendo um ponto a considerar nas novas funcionalidades a implementar e que será descrito com mais detalhe na secção seguinte.

No que respeita à aplicação correspondeu às expectativas do cliente, cumprindo com o que ele pediu. Ao longo do projecto foi lhe apresentada a evolução do produto o que permitiu ir melhorando alguns aspectos, sem nunca esquecermos que a aplicação tem potencial para crescer.

A aplicação vai permitir que os utilizadores consigam consultar toda a informação referente ao clube, tendo a possibilidade de consultar a informação sobre o jogo, analisar o jogo em si, quem é que foram os seus intervenientes ou que eventos aconteceram ao longo do jogo. Foi ainda concebida a hipótese de os mesmos poderem analisar a *performance* da equipa, jogadores e treinadores, ponto este que se notou que faltava nos *websites* referentes aos principais clubes de Portugal. Assim, os utilizadores desta aplicação poderão analisar esses dados.

O desenvolvimento desta aplicação permitiu explorar diversas tecnologias e melhorar o conhecimento sobre as tecnologias *web*. Foi também fundamental entender e utilizar um

processo para o desenvolvimento de *software*, e usar algumas técnicas importantes para desenvolver uma aplicação robusta e segura.

Em suma, e tendo em conta o desenvolvimento da aplicação, considera-se que os objectivos da dissertação foram alcançados. No entanto existe uma vontade de continuar a melhorar aplicação quer a nível funcional, quer a nível de *design*. Como se sabe, um produto tem que estar sempre na vanguarda das novas tecnologias e com isso é necessário analisar as tecnologias que podem ser importantes para o produto.

6.2 TRABALHO FUTURO

Nesta secção vão ser abordados alguns aspectos que podem ser implementados na aplicação, de forma a sustentar o seu crescimento e melhorar a mesma, uma vez que as tecnologias estão em constante mutabilidade.

Um dos pontos importantes é a concepção de uma aplicação *mobile*. Para colmatar esta inexistência, a aplicação *web* foi desenvolvida de forma a adaptar-se a qualquer dispositivo. Para isso já foram implementados *web services*, sendo posteriormente mais fácil conceber a aplicação *mobile*, fazendo com que a mesma fique apenas responsável por comunicar com o produto implementado.

No que respeita ao *design* da aplicação, foi desenvolvido um *layout* minimalista, dando a possibilidade a cada clube de adaptar o *frontend*, nomeadamente quanto às cores e símbolos, entre outras características específicas de cada clube desportivo. É necessário salientar que esta aplicação foi desenvolvida a pensar também noutros desportos, ou seja, o clube apenas tem que mudar os seus eventos associados aos jogos. Posteriormente, terá que ser adaptada a informação que surge no *frontend*, mas a nível de *backend* e de gerir informação, não existem novas funcionalidades. Esta abordagem de desenvolvimento permite a adaptabilidade a outros desportos.

No que concerne à integração com outros produtos ou parcerias, também poderá ser implementada a venda ou disponibilização de informação. Para tal efeito, será disponibilizada uma *API-key* de forma a que as aplicações se possam identificar para consumir informação.

Vai ainda ser implementado um mecanismo de importação de dados, para que os novos clubes consigam carregar a informação mais facilmente, sendo no entanto necessário, procurar parcerias para tal. Será interessante utilizar o mecanismo de *cronjobs*, ou seja, implementar rotinas para decorrer num determinado tempo, por exemplo a todas as terças-feiras vai ser realizada uma chamada a um *web service* de um parceiro para ir buscar informação de um jogo ou da tabela classificativa.

Em relação aos utilizadores, é necessário oferecer-lhes mais interactividade com a aplicação em si, permitindo ao mesmo inserir comentários sobre um jogo, ou até mesmo inserir informação sobre a história do clube. Foi implementado um sistema de *login* para os

utilizadores, com o intuito de oferecer esta aproximação dos utilizadores da aplicação no futuro. No entanto toda essa informação será visionada pelo administrador da aplicação para garantir a confiabilidade da informação.

Com as demonstrações ao cliente, verificou-se que aplicação estava funcional e dentro das expectativas do mesmo. Contudo, com a interacção com a aplicação, o cliente solicitou algumas melhorias que vão ser realizadas para que o mesmo tenha um manuseamento mais agradável com a aplicação.

Ao longo da dissertação e da implementação do produto, foram surgindo novas ideias e potencialidades que no início não se tinham percepcionado. Uma das ideias possíveis era distribuir a aplicação a uma associação, por exemplo Associação de Futebol de Braga. Cada clube iria ter uma aplicação onde guardaria a informação sobre os seus jogadores e treinadores. No entanto, a Associação teria a informação de todos os clubes que vai consultar aos *web services* e com isso cada equipa de arbitragem teria de utilizar a aplicação para inserir a informação sobre o jogo. Ao fim de inserir a informação do jogo, essa informação será guardada na aplicação da Associação e dos respectivos clubes. Esta abordagem seria interessante para centralizar a informação, reduzir a preocupação dos dirigentes e dos árbitros, sendo necessário salientar que era uma excelente abordagem para os clubes não profissionais conseguirem seguir o crescimento da sua equipa e respectivos elementos.

BIBLIOGRAFIA

[1] Citador, "Confúcio." http://www.citador.pt/frases/escolhe-um-trabalho-de-que-gostes-e-nao-teras-qu-confucio-9953, 2017. [Online; acedido em 24-Setembro-2017].

[2] J. M. Fernandes and R. J. Machado, *Requirements in Engineering Projects*. Springer, 1st ed., 2016. ISBN 978-3-319-18596-5.

[3] R. Pressman, *Software Engineering A Practitioner's Approach*. McGraw-Hill, 7th ed., 2010. ISBN 978–0–07–337597–7.

[4] R. Nixon, *Learning PHP, MySQL, JavaScript, and CSS,*, ch. 9: Mastering MySQL, p. 216. O'Reilly Media, 2th ed., 2012. ISBN: 978-1-449-31926-7.

[5] T. O'reilly, "What is web 2.0: Design patterns and business models for the next generation of software," *Communications & strategies*, 2007. http://www-public.tem-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/OReilly07.pdf ([Online; acedido em 24-Outubro-2017]).

[6] w3techs, "Usage statistics and market share of web servers for websites, october 2017." https://w3techs.com/technologies/overview/web_server/all, 2017. [Online; acedido em 14-Outubro-2017].

[7] Phalcon, "The mvc architecture." https://docs.phalconphp.com/en/latest/reference/mvc.html, 2017. [Online; acedido em 24-Novembro-2015].

[8] Developer.chrome, "Mvc architecture." https://developer.chrome.com/apps/app_frameworks, 2015. [Online; acedido em 24-Outubro-2017].

[9] w3techs, "Usage statistics and market share of server-side programming languages for websites, october 2017." https://w3techs.com/technologies/overview/programming_language/all, 2017. [Online; acedido em 14-Outubro-2017].

[10] DB-Engines, "Db-engines ranking." https://db-engines.com/en/ranking, 2017. [Online; acedido em 26-Setembro-2017].

[11] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High Performance MySQL*, ch. 1 1: MySQL Architecture and History, p. 2. O'Reilly Media, 3th ed., 2012. ISBN 978–1–449–31428–6.

[12] A. Lopez, *Learning PHP 7*, ch. 8: Using Existing PHP Frameworks, p. 416. Packt Publishing ltd, 1st ed., 2016. ISBN 978-1-78588-054-4.

[13] OWASP, "About the open web application security project." https://www.owasp.org/index.php/About_OWASP, 2017. [Online; acedido em 22-Março-2017].

[14] OWASP, "Sql injection prevention cheat sheet." https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet, 2017. [Online; acedido em 22-Março-2017].

[15] OWASP, "Cross-site request forgery (csrf) - owasp." https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF), 2017. [Online; acedido em 22-Março-2017].

[16] L. J. Mitchell, *PHP Web Services*, ch. Preface, p. 8. O'Reilly Media, 2th ed., 2016. ISBN 978-1-491-93309-1.

[17] T. P. Group, "Php: Introduction - manual." http://php.net/manual/en/intro.mysql.php, 2017. [Online; acedido em 10-Agosto-2017].

[18] Phalcon, "Working with models — phalcon 3.1.1 documentation (english)." https://olddocs.phalconphp.com/en/3.0.0/reference/models.html, 2017. [Online; acedido em 30-Outubro-2016].

[19] Phalcon, "Hello world benchmark." https://docs.phalconphp.com/en/1.2.6/reference/benchmark/hello-world.html, 2017. [Online; acedido em 15-Agosto-2017].

[20] Phalcon, "General phalcon framework." https://docs.phalconphp.com/en, 2015. [Online; acedido em 30-Outubro-2016].

[21] D. Schissler and S. Iakovlev, *Phalcon Cookbook*. Packt Publishing ltd, 1st ed., 2016. ISBN 978-1-78439-688-6.

[22] A. Hussain, *Learning PHP 7 High Performance*. McGraw-Hill, 1st ed., 2016. ISBN 978-1-78588-226-5.

[23] C. Snyder, T. Myer, and M. Southwell, *Pro PHP Security*. Apress, 2th ed., 2010. ISBN 978-1-4302-3318-3.

[24] B. Edmunds, *Securing PHP Apps*. Apress, 2th ed., 2016. ISBN 978-1-4842-2120-4.

[25] T. Ballad and W. Ballad, *Securing PHP web applications*. Pearson Education, Inc, 2th ed., 2009. ISBN 978-0-321-53434-7.

[26] Xdebug, "Xdebug: Documentation." https://xdebug.org/docs/remote, 2016. [Online; acedido em 20-Março-2017].

[27] E. Westby, *Git for teams*. O'Reilly Media, 1st ed., 2015. ISBN 978-1-491-91118-1.

[28] PHPUnit, "Phpunit manual." https://phpunit.de/manual/current/en/index.html, 2015. [Online; acedido em 20-Fevereiro-2017].

A

CARTÃO *VOLERE*

Todas as funcionalidades do projecto, foram definidas através do cartão de *volere*, o que permitiu especificar diversas informações, não sendo só o requisito em si. Seguidamente serão apresentados todos os requisitos funcionais e não funcionais referentes a aplicação.

Requisito:	1	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir Jogador, Editar jogador, Consultar jogador, Eliminar jogador
Descrição:	O administrador insere/edita/consulta/elimina detalhes de um jogador				
Fundamentação:	Para que a informação de um determinado jogador possa estar sempre correta.				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue, em qualquer altura, visualizar, consultar, editar e excluir os dados do jogador				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	2	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar jogador
Descrição:	O administrador procura um jogador				
Fundamentação:	Possibilita ao administrador uma melhor facilidade de encontrar determinado jogador				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador consegue encontrar facilmente um jogador, introduzindo informação que identifique o mesmo				
Satisfação do Utilizador:	4		**Insatisfação do utilizador:**		
Prioridade:	Baixa		**Conflitos:**	Não	
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	3	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir jogo, Editar jogo, Consultar jogo, Eliminar jogo
Descrição:	O administrador insere/edita/consulta/elimina informação sobre os jogos associados à equipa				
Fundamentação:	Para que a informação de um determinado jogo esteja sempre correta				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue, em qualquer altura, visualizar, editar, consultar e excluir os dados referente a um determinado jogo				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	4	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar jogo
Descrição:	O administrador pesquisa jogos do clube				
Fundamentação:	Para que o administrador tenha uma maior facilidade em encontrar determinado jogo				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador consegue encontrar determinado jogo inserido um detalhe do mesmo, permitindo assim uma maior rapidez na ação				
Satisfação do Utilizador:	4	**Insatisfação do utilizador:**			
Prioridade:	Baixa	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	5	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir evento, Editar evento, Consultar evento, Eliminar evento
Descrição:	O administrador insere/edita/lista/elimina os eventos associados ao jogo				
Fundamentação:	Permite que o administrado coloque informação importante referente a um determinado jogo				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador regista a informação sobre o jogo e depois os utilizadores podem consultar a mesma				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	6	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar eventos do jogo
Descrição:	O administrador pesquisa eventos				
Fundamentação:	Permite que o administrador encontre determinado evento				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador insere o nome do evento e surgem os detalhes sobre o evento que procurou				
Satisfação do Utilizador:	3		**Insatisfação do utilizador:**		
Prioridade:	Baixa		**Conflitos:**	Não	
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	7	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir estatísticas, Editar estatísticas, Consultar estatísticas
Descrição:	O administrador insere/edita/consulta todos as estatísticas relativas ao jogo				
Fundamentação:	Para que as estatísticas de um determinado jogo possam estar fidedignas				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue, em qualquer altura, visualizar, editar e consultar as estatísticas de um determinado jogo				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	8	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir árbitro, Editar árbitro, Consultar árbitro, Eliminar árbitro
Descrição:	O administrador regista/altera/consulta/remove informação de um árbitro				
Fundamentação:	Permite registar todos os árbitros, para facilitar o registo dos eventos associados aos jogos.				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue, em qualquer altura, visualizar, editar e consultar informação sobre árbitros.				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	9	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar árbitro
Descrição:	O administrador pesquisa árbitros				
Fundamentação:	Permite que o administrador encontre determinado árbitro				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador insere o nome de um determinado árbitro e surgem os detalhes sobre o árbitro que procurou				
Satisfação do Utilizador:	3	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	10	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir estádio, Editar estádio, Consultar estádio, Eliminar estádio
Descrição:	O administrador regista/altera/consulta/remove informação de um estádio				
Fundamentação:	Para que a informação de um determinado estádio possa ser inserida e atualizada a qualquer momento.				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue, em qualquer altura, visualizar, editar e excluir os dados do estádio				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	11	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar estádio
Descrição:	O administrador procura estádios				
Fundamentação:	Permite ao administrador encontrar determinado estádio com maior rapidez				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador deve inserir detalhes sobre o árbitro e depois surgirá informação sobre o estádio				
Satisfação do Utilizador:	3	**Insatisfação do utilizador:**			
Prioridade:	Baixa	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	12	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir equipa, Editar equipa, Consultar equipa, Eliminar equipa
Descrição:	O administrador insere/altera/lista/elimina as equipas adversárias				
Fundamentação:	Permite que os detalhes sobre as equipas adversárias sejam manipulados com facilidade pelo administrador				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador tem a possibilidade de inserir, editar, atualizar ou eliminar a informação sobre uma equipa adversária.				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	13	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar equipa adversária
Descrição:	O administrador pesquisa as equipas adversárias				
Fundamentação:	Permite que os detalhes sobre as equipas adversárias sejam alterados com facilidade pelo administrador				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador tem a possibilidade de inserir, editar, atualizar ou eliminar a informação sobre uma equipa adversária.				
Satisfação do Utilizador:	3	**Insatisfação do utilizador:**			
Prioridade:	Baixa	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	14	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir treinador, Editar treinador, Consultar treinador, Eliminar treinador
Descrição:	O administrador regista/altera/consulta/elimina informação do treinador				
Fundamentação:	Permite que os detalhes sobre os treinadores sejam manipulados com facilidade pelo administrador				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador tem a possibilidade de inserir, editar, atualizar ou eliminar a informação sobre um treinador				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente– 18.10.2016				

Requisito:	15	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar treinador
Descrição:	O administrador pesquisa treinador				
Fundamentação:	Possibilita ao administrador uma filtragem do treinador que procura				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador deve digitar um detalhe sobre o treinador, de modo a surgir a informação sobre o mesmo				
Satisfação do Utilizador:	3	**Insatisfação do utilizador:**			
Prioridade:	Baixa	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	16	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir competição, Editar competição, Consultar competição, Eliminar competição
Descrição:	O administrador insere/edita/elimina as competições que a equipa disputa				
Fundamentação:	Permite criar as competições em que a equipa participa numa dada época. Os jogos disputados estarão associados a uma dada competição.				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador insere, edita e elimina todas as competições, para que depois os jogos sejam associados a uma determinada competição				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	17	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar competição
Descrição:	O administrador procura as competições				
Fundamentação:	Permitir que o administrador encontre determinada competição de uma forma mais eficaz				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador deve definir um detalhe que ajude o sistema a devolver-lhe a competição que procura				
Satisfação do Utilizador:	3	**Insatisfação do utilizador:**			
Prioridade:	Baixa	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	18	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir época, Editar época, Consultar época, Eliminar época
Descrição:	O administrador insere/altera/consulta/remove informação referente a época desportiva				
Fundamentação:	Permite que o administrador gere toda informação sobre uma época desportiva				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue, em qualquer altura, visualizar, editar e consultar informação sobre uma época desportiva.				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	19	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar época
Descrição:	O administrador pesquisa época desportiva				
Fundamentação:	Para que o administrador consiga encontrar determinada época desportiva				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador deve introduzir um detalhe sobre a época que procura e posteriormente terá os dados sobre a época que procura				
Satisfação do Utilizador:	3	**Insatisfação do utilizador:**			
Prioridade:	Baixa	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	20	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir notícia, Editar notícia, Consultar notícia, Eliminar notícia
Descrição:	O administrador cria/edita/consulta/elimina notícias;				
Fundamentação:	Permite que o administrador controle a informação sobre as notícias que são publicadas				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue introduzir, alterar, eliminar e consultar informação sobre as notícias do clube				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	21	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisar notícia
Descrição:	O administrador procura notícias				
Fundamentação:	Permite ao administrador encontrar determinada notícia para que depois faça a gestão da sua informação				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador deve introduzir uma informação sobre a notícia conseguindo apresentar a mesma				
Satisfação do Utilizador:	3	**Insatisfação do utilizador:**			
Prioridade:	Baixa	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	22	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir história, Editar história, Eliminar história e consultar história
Descrição:	O administrador insere/edita/consulta/elimina eventos históricos associado a equipa				
Fundamentação:	O administrador tem o controlo total sobre as histórias que são publicadas no website				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador consegue introduzir, alterar, eliminar e consultar informação sobre a história do clube				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	23	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Pesquisa história
Descrição:	O administrador consulta eventos históricos associados a equipa				
Fundamentação:	Para que o administrador consiga encontrar determinado evento histórico.				
Origem/Autor:	Cliente				
Critério de Ajuste:	O administrador ao pesquisar determinada notícia com os parâmetros inseridos pelo mesmo, terão uma rápida pesquisa.				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	24	Tipo de Requisito:	Funcional	Evento/Caso de uso:	Editar dados pessoais
Descrição:	O administrador insere/edita os dados pessoais				
Fundamentação:	Permite ao administrador editar os seus dados pessoais				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador digita a informação pessoal e depois serão guardados esses novos dados pessoais				
Satisfação do Utilizador:	5		Insatisfação do utilizador:		
Prioridade:	Alta		Conflitos:	Não	
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	25	Tipo de Requisito:	Funcional	Evento/Caso de uso:	Acesso ao BackOffice
Descrição:	O administrador autentica-se na aplicação				
Fundamentação:	Permite ao administrador autenticar-se na aplicação para entrar no seu perfil e aceder as suas permissões e dados.				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O administrador tem um formulário onde tem de realizar a sua autenticação para aceder a aplicação				
Satisfação do Utilizador:	4		Insatisfação do utilizador:		
Prioridade:	Alta		Conflitos:	Não	
Materiais de Apoio:	Análise de segurança da aplicação				
Histórico:	V1.0: Análise final dos requisitos – 10.02.2017				

Requisito:	26	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consultar jogo
Descrição:	O utilizador consulta informações sobre determinado jogo				
Fundamentação:	Para que a informação de um determinado jogo possa consultada pelo utilizador				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador consegue, em qualquer altura, visualizar informações sobre os jogos.				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015 V2.0: Correção com o cliente – 18.10.2016				

Requisito:	27	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consultar estatísticas de jogadores
Descrição:	O utilizador consulta estatísticas sobre os jogadores.				
Fundamentação:	Possibilita ao utilizador consultar as estatísticas referentes a um jogador				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador seleciona um jogador e posteriormente visualiza as estáticas do jogador.				
Satisfação do Utilizador:	4		**Insatisfação do utilizador:**		
Prioridade:	Média		**Conflitos:**	Não	
Materiais de Apoio:	Análise de melhorias da aplicação				
Histórico:	V1.0: Melhorias da aplicação– 20.02.2017				

Requisito:	28	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consulta confrontos de jogadores
Descrição:	O utilizador consulta estatísticas de confrontos de jogadores com equipas adversárias				
Fundamentação:	Possibilita ao utilizador consultar as estatísticas referentes a um jogador contra uma determinada equipa numa determinada competição				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador seleciona um jogador e posteriormente visualiza as estáticas do confronto que escolheu				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	29	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consulta confronto de equipas
Descrição:	O utilizador consulta estatísticas de confrontos com equipas adversárias				
Fundamentação:	Possibilita ao utilizador consultar as estatísticas referentes a confrontos com equipas adversárias				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador seleciona uma equipa adversária e posteriormente visualiza as estáticas do confronto				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	30	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Inserir/Editar dados pessoais
Descrição:	O utilizador insere/edita os dados pessoais				
Fundamentação:	Permite ao utilizador analisar o desempenho de um jogador perante jogadores e equipas adversárias.				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador tem de escolher um jogador, para depois comparar com um jogador ou uma equipa que pretender comparar.				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Análise da usabilidade da aplicação – 20.01.2017				

Requisito:	31	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Visualiza informação de jogador
Descrição:	O utilizador visualiza dados sobre jogadores em determinada época				
Fundamentação:	Permite ao utilizador analisar o desempenho de um jogador perante jogadores e equipas adversárias.				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador tem de escolher um jogador, para depois comparar com um jogador ou uma equipa que pretender comparar.				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	32	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consulta desempenho da equipa
Descrição:	O utilizador consulta o desempenho da equipa em determinada competição				
Fundamentação:	Permite ao utilizador analisar o desempenho da equipa em determinada competição				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador seleciona uma competição e depois é apresentado o desempenho da equipa naquela competição em determinada época				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	33	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Registo na aplicação
Descrição:	O utilizador regista-se na aplicação				
Fundamentação:	Permite ao utilizador registar-se na aplicação de modo a aceder a mesma				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador tem um formulário onde tem de realizar o seu registo para aceder a aplicação.				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	34	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Login
Descrição:	O utilizador autentica-se na aplicação				
Fundamentação:	Permite ao utilizador autenticar-se na aplicação para entrar no seu perfil e aceder as suas permissões e dados.				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador tem um formulário onde tem de realizar a sua autenticação para aceder a aplicação.				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	35	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Login via Facebook
Descrição:	O utilizador utiliza a conta do Facebook para se autenticar				
Fundamentação:	Permite ao utilizador aceder a aplicação com a sua conta de Facebook				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador tem a possibilidade de se conectar com a sua conta de Facebook				
Satisfação do Utilizador:	4	**Insatisfação do utilizador:**			
Prioridade:	Média	**Conflitos:**	Não		
Materiais de Apoio:	Análise de melhorias da aplicação				
Histórico:	V1.0: Melhorias da aplicação– 20.02.2017				

Requisito:	36	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Recuperação de password
Descrição:	O utilizador recupera a password				
Fundamentação:	Permite ao utilizador recuperar a sua password, caso se esqueça da mesma				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador irá solicitar uma nova password e seguidamente será enviado um email com a nova password				
Satisfação do Utilizador:	4	**Insatisfação do utilizador:**			
Prioridade:	Média	**Conflitos:**	Não		
Materiais de Apoio:	Análise de melhorias da aplicação				
Histórico:	V1.0: Melhorias da aplicação– 20.02.2017				

Requisito:	37	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consultar notícias
Descrição:	O utilizador consulta notícias do clube				
Fundamentação:	Permite ao utilizador ver as notícias que estão associadas a equipa				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador pretende ver as noticias do seu clube e com esta opção consegue ver o que se passa com a sua equipa				
Satisfação do Utilizador:	5	**Insatisfação do utilizador:**			
Prioridade:	Alta	**Conflitos:**	Não		
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	38	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Consultar histórias
Descrição:	O utilizador consulta histórias do clube				
Fundamentação:	Permite ao utilizador ver a história associada ao seu clube				
Origem/Autor:	Cliente				
Critério de Ajuste:	O utilizador pretende consultar a história da sua equipa de modo a ficar contextualizado com os antepassados da sua equipa				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Reuniões				
Histórico:	V1.0: Criação do requisito – 28.11.2015				

Requisito:	39	**Tipo de Requisito:**	Funcional	**Evento/Caso de uso:**	Validar login
Descrição:	O sistema valida o login				
Fundamentação:	Permite aos utilizadores saberem se tem acesso a aplicação				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	Os utilizadores inserem as suas credencias, caso o login seja válido tem acesso a determinadas ações/BackOffice				
Satisfação do Utilizador:	4		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Análise de segurança da aplicação				
Histórico:	V1.0: Análise final dos requisitos – 10.02.2017				

Requisito:	40	**Tipo de Requisito:**	Não-Funcional	**Evento/Caso de uso:**	
Descrição:	A interface da aplicação web adapta-se a qualquer dispositivo				
Fundamentação:	Permite ao utilizador aceder à aplicação web com qualquer dispositivo				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador acede à aplicação web com um dispositivo (Tablet, smartphone e PC) e consegue navegar no mesmo sem qualquer restrição.				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Análise inicial da aplicação				
Histórico:	V1.0: Melhorias da aplicação– 20.01.2017				

Requisito:	41	**Tipo de Requisito:**	Não-Funcional	**Evento/Caso de uso:**	
Descrição:	A aplicação web oferece usabilidade aos utilizadores				
Fundamentação:	O utilizador tem a possibilidade de ter uma interação intuitiva com o website				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador ao utilizar a aplicação web vai sentir uma boa dinâmica com a mesma devido a sua usabilidade				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Análise de usabilidade				
Histórico:	V1.0: Melhorias da aplicação– 20.01.2017				

Requisito:	42	**Tipo de Requisito:**	Não-Funcional	**Evento/Caso de uso:**	
Descrição:	A aplicação web deverá garantir que não existem acessos não autorizados ao backend				
Fundamentação:	Os utilizadores sem acesso ao backend não poderão aceder ao mesmo				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador ao aceder ao backend se não tiver as credenciais corretas não conseguirá aceder ao mesmo				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Análise de segurança da aplicação				
Histórico:	V1.0: Análise final dos requisitos – 10.02.2017				

Requisito:	43	**Tipo de Requisito:**	Não-Funcional	**Evento/Caso de uso:**	
Descrição:	A aplicação web garante escalabilidade.				
Fundamentação:	Permite que vários utilizadores estejam a aceder à aplicação web.				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	Quando existir vários utilizadores a aceder à aplicação web, vão estar precavidos em relação ao problema de escalabilidade da aplicação.				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Análise inicial da aplicação				
Histórico:	V1.0: Melhorias da aplicação – 15.10.2015				

Requisito:	44	**Tipo de Requisito:**	Não-Funcional	**Evento/Caso de uso:**	
Descrição:	A aplicação web oferece um tempo de resposta curto aos utilizadores.				
Fundamentação:	Oferecer ao utilizador a informação solicitada por ele de uma forma rápida e coerente.				
Origem/Autor:	Henrique Sobral, Programador				
Critério de Ajuste:	O utilizador solicita uma informação à aplicação web e a mesma responde num período de tempo reduzido.				
Satisfação do Utilizador:	5		**Insatisfação do utilizador:**		
Prioridade:	Alta		**Conflitos:**	Não	
Materiais de Apoio:	Análise inicial da aplicação				
Histórico:	V1.0: Melhorias da aplicação – 15.10.2015				